# Query Lookahead for Query-Based Document Categorization

by

Bienvenido José Vélez–Rivera

B.S., Cornell University (1986)
M.S., University of California Berkeley (1988)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

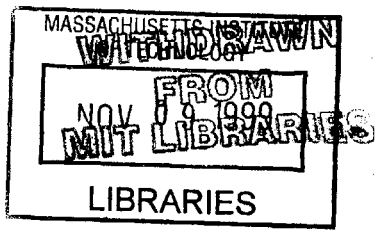
Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1999






Signature of Author .....
Department of Electrical Engineering and Computer Science
September 7, 1999

Certified by .....
David K. Gifford
Professor of Computer Science
Thesis Supervisor

Accepted by .....
Arthur Smith
Chairman, Departmental Committee on Graduate Students

# Query Lookahead for Query-Based Document Categorization

by

Bienvenido José Vélez–Rivera

Submitted to the Department of Electrical Engineering and
Computer Science
on August 10, 1999, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

## Abstract

The central thesis of this dissertation is that *query lookahead* can be used to improve the effectiveness of text-based information retrieval systems. Query lookahead is the process of eagerly evaluating multiple refined queries that are automatically derived from an initial seed query. We introduce *LOOK*, an efficient query lookahead algorithm showing that the technique can be integrated with previous query refinement algorithms with constant overhead.

LOOK can be used to explore how terms categorize a large result set. Thus it can also be used to select terms for refining a query. A *coverage term selection algorithm* processes the results of LOOK to select a set of queries that succinctly categorizes a query result set. We develop a mathematical model of ideal coverage term selection and use it to prove that the problem is NP-complete. In response, we develop *CTS*, a greedy approximation algorithm, and demonstrate that coverage term selection can improve the quality of terms when compared to previous frequency-based algorithms.

Finally, we introduce *interactive query hierarchies*, a new type of user interface for result set visualization. The new interface organizes a result set into a hierarchy of categories denoted by queries. The main idea is to label categories of documents with query fragments that enable the user and the system to engage in a dialog in a single consistent language framework. *LOOK* serves as the underlying mechanism for populating categories with documents.

Thesis Supervisor: David K. Gifford
Title: Professor of Computer Science

## Acknowledgments

For me, a product of the Puerto Rican public school system, the journey through an MIT PhD was not easy. During the six arduous years there were many times when I got very close to giving up. Fortunately, there were also people and forces that kept me going.

A graduate student's first line of support is its research group. Professor David K. Gifford, my thesis advisor and our group leader, firmly stood by me during some of my toughest times at the lab. I spent many enjoyable times working with my graduate student friends Ron Weiss and Mark Sheldon. I could safely say that I learned more about the inner nature of research by working with Ron and Mark than by any other means. Jeanne Darling, our group's administrative assistant, gave me her unconditional support and kindhearted company whenever I needed it. Alex Hartemink, Julia Khodor and Qian Z. Wang suggested many improvements to both my dissertation and my thesis defense talk.

My thesis committee members, David Karger and Barbara Liskov provided many important suggestions for improving this dissertation. Professor Albert Meyer, my academic advisor at MIT, insightfully guided me through the intricacies of the graduate program. I will never forget the critical encouragement that I received from Professor Manuel Blum, both through his acts as my student mentor as well as through his example. Manuel is living proof that one can be both a great scientist and an outstanding human being. Pedro I. Rivera helped me with the NP-completeness proof in Chapter 3.

My wife Melba Roque bravely and lovingly withstood the demands of getting accustomed to a place and culture foreign to us while having a husband too often distracted on his work. I will eternally be grateful for her perseverance and loving support. My daughter Cristina and my sons Eduardo and Ricardo were a constant source of inspiration. My parents Bienvenido Vélez and Ana J. Rivera always encouraged me to fulfill my dreams. Last but not least, my desire to give my best to my beloved Puerto Rico was ever present during my graduate career.

TO MELBA, CRISTINA, EDUARDO, AND RICARDO
WITH ALL MY LOVE

This thesis describes research done at the Laboratory for Computer Science at the Massachusetts Institute of Technology, supported by the Defense Advanced Research Projects Agency of the Department of Defense contract number DABT63-95-C-0005.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 The Thesis

The central thesis of this dissertation is that *query lookahead* can be used to improve the effectiveness of text-based information retrieval systems. Query lookahead is the process of eagerly evaluating multiple queries that are automatically derived from an initial seed query. In particular, we demonstrate that the information about such eagerly computed result sets can be used to improve term selection. Using query lookahead, we developed *interactive query hierarchies*, a new type of user interface for result set visualization. The new interface organizes a result set into a hierarchy of categories denoted by queries. The main idea is to label categories of documents with query fragments that enable the user and the system to engage in a dialog in a single consistent language framework.

## 1.2 The Problem

The amount of textual information available online via the World Wide Web is rapidly approaching a terabyte of data. In 1997, the AltaVista search engine was reported to be indexing one hundred million web documents with an average size of five kilobytes of text per document [9]. In the same year, the search engine reported handling twenty-six

million user queries on an average day. This explosion in information storage and access is placing unprecedented demands on information retrieval technology. This dissertation presents new techniques to aid with the problem of finding information in such large information repositories.

An ideal information retrieval system interprets input that represents a human information need and computes the most relevant subset of information from a corpus of documents. Thus, an information retrieval system must provide a query language for expressing information needs, a means for representing the information database in digital form, and a means for comparing the expression of the information need with the information available in the database.

The level of specificity of a user's information need plays an important role in determining the most effective approach to satisfy the need. A user may have a very specific need that can be translated into a written question or query. This type of need leads naturally to a *searching* approach. Searching often involves the process of specifying and evaluating a query against a database. In contrast to a specific need, a user may have little knowledge about what information might be available and therefore may not be able to specify a precise query. This second type of need is related to *browsing* a collection. Browsing organizes information so that a user can decide what is of interest. A good example is a user who wishes to examine the news of the day. The user often cannot predict what happened and thus a query must necessarily be general in scope and specific to a date. Browsing and searching are not individually adequate for all needs and therefore information retrieval (IR) systems should integrate both services in a coherent fashion.

## 1.3 The Solution

Underlying the contributions of this dissertation are two user interaction paradigms that combine searching and browsing in a way that increases the synergy between both ser-

vices: query refinement and interactive query hierarchies. The central idea is to offer the user assistance with the query formulation process by exploiting combined knowledge about the available information with knowledge about a user's information need as encoded by an initial *seed* query. In both cases, the assistance is in the form of automatically generated suggestions for improving the seed query. These suggestions can take the form of terms that a user can manually combine with the seed query or entire new queries automatically constructed by combining new terms with the seed query.

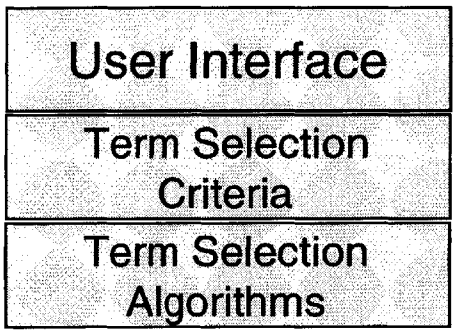



Figure 1-1: Abstraction levels at which this dissertation makes contributions

Our contributions can be conveniently organized in terms of the three levels of abstraction depicted in figure 1-1: the user interface level, the term selection criteria level, and the term selection algorithms level. The user interface level deals with the way in which suggestions are displayed to a user as well as the means provided to a user for manipulating such suggestions in order to focus the retrieval process. The term selection criteria level establishes the goals behind the process of selecting which terms should be used to generate suggestions. The term selection algorithms level deals with the methodology employed to select term suggestions that satisfy the goals set forth by the term selection criteria level. Using this three level model we will now describe our main contributions.

### 1.3.1 Frequency-based Query Refinement

Query refinement, the focus of Chapter 2, recommends automatically generated term suggestions that a user can employ as a guide for improving the seed query. At the user interface level, Chapter 2 proposes a query refinement system that, in response to a seed query, automatically displays a selection of terms appearing in documents that match the seed query. A user can select terms that appear to be related to his/her information need and combine these terms with the seed query using logical operations. The user can add suggested terms to the seed query conjunctively to reduce the result set size, disjunctively to broaden the scope of the query, by negation to focus the query away from a particular topic, or in lieu of existing query terms to modify the query.

Figure 1-2 shows refinement suggestions for the seed query `text:scheme` (a programming language developed at MIT) generated by the HyPursuit prototype search tool[68]. As in the Semantic File System [26], HyPursuit allows attributes to be specified with terms to represent the portion of the document where the term should appear. In the example in Figure 1-2 attribute `text` indicates that the term `scheme` may appear anywhere inside a matching document.

At the term selection criteria level, Chapter 2 proposes two complementary experimental metrics, called *Concept Recall* and *Precision Improvement*, that can be used to assess the effectiveness of term selection algorithms. Concept recall measures the ability of a term selection algorithm to generate terms that are semantically related to the seed query. Precision improvement measures the ability of an algorithm to select terms that improve the precision of a query. Both of our metrics are unique in their focus on single-term query modifications and on measurements that take into account initial query precision. Our focus on single-term query modifications is of particular importance given that we focus our experiments on refinement of short seed queries. We have experimentally observed that short queries can easily diverge from their originally intended meaning when they are expanded with multiple terms. Assessing the effectiveness of query refinement considering initial precision is important since the difficulty of refining

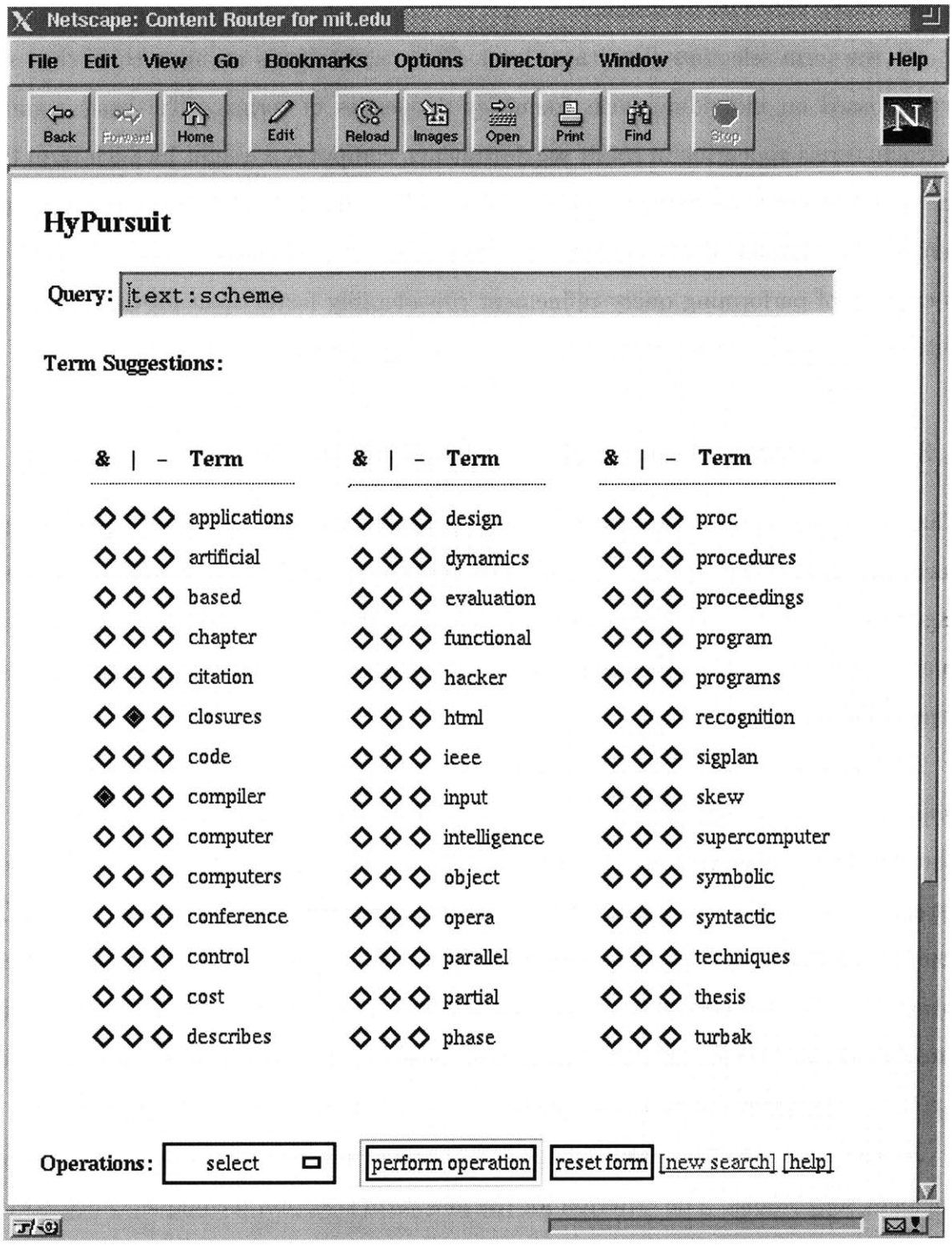



Figure 1-2: Query refinement output generated by the HyPursuit network search engine in response to the seed query `text:scheme`

a seed query varies its precision.

At the term selection algorithms level, Chapter 2 focuses on algorithms that select terms based on individual term frequency properties of terms. The basic algorithm extracts terms appearing in result set documents, computes a weight for each term based on how often the term appears in these documents, and finally picks the highest weighted terms. The chapter also introduces and analyzes a new algorithm, called RMAP, that is capable of performing query refinement considerably faster than previous algorithms with a small degradation in both concept recall and precision improvement.

### 1.3.2 Coverage Query Refinement using Query Lookahead

Chapter 3 is mostly concerned with the term selection criteria and term selection algorithms abstraction levels. This chapter introduces *query lookahead*, the process of eagerly evaluating multiple refined queries automatically generated from an initial seed query. This new technique is designed to capitalize on expected future improvements in computing capacity.

At first sight it may seem that eager query evaluation of many queries may increase response time to intolerable levels. Chapter 3 introduces a new algorithm called *LOOK* that simultaneously extracts terms from result set documents and keeps track of the result set generated by each extracted term when conjoined with the seed query. Thus, *LOOK* combines term extraction with query lookahead. The algorithm essentially computes an inverted file for the documents matching the seed query. Using *LOOK* we demonstrate theoretically and empirically that query lookahead can be achieved at a cost that is within a constant factor of the cost of extracting terms from result set documents.

At the term selection level, Chapter 3 develops new selection criteria for query refinement suggestions. The criterion for the new term selection technique, called *coverage term selection*, is to minimize the information loss that may result from suggesting terms that do not expose significant portions of the seed query's result set without increasing the redundancy resulting from terms that convey the same information. The idea is to

select terms whose corresponding refined queries generate result sets that approximate a balanced partition of the seed query.

At the term selection algorithms level, Chapter 3 demonstrates that selecting terms that simultaneously minimize redundancy and information loss is an NP-complete problem. The chapter then proceeds to demonstrate that a simple greedy algorithm, called *CTS*, is capable of achieving both lower redundancy and lower information loss than the frequency-based algorithm that demonstrated the best performance in Chapter 2.

### 1.3.3 Interactive Query Hierarchies for Result Set Visualization

In Chapter 4, we present a new user interface integrating searching and browsing into a single coherent user interface. We call this approach *interactive query hierarchies.* The central idea is to organize a result set in a hierarchy using queries as the basis for forming the various categories upon which the documents are organized. In fact, the same query language used by the users to describe their information needs is used by the system to describe its information resources to the user. One advantage of this approach is that categories inherit their semantics from queries. For instance, the categories that a given document belongs to are unambiguously determined by the semantics of the query language used. Documents belong to all the categories (queries) that match the document. A second advantage is that queries serve as extremely compact category descriptors.

In response to a query, the system generates refined queries by extracting terms extracted from the documents in the result set and combining these terms with the seed query. The system then eagerly determines the result set of each refined query using query lookahead. The hierarchical view of the result set not only provides an organized and succinct view of the available choices for refining the seed query, but also allows the user to examine the result set of each refined query instantaneously.

The screen snapshot in Figure 1-3 shows a user interface that implements interactive query hierarchies. The snapshot was taken from the *Inforadar* network search engine

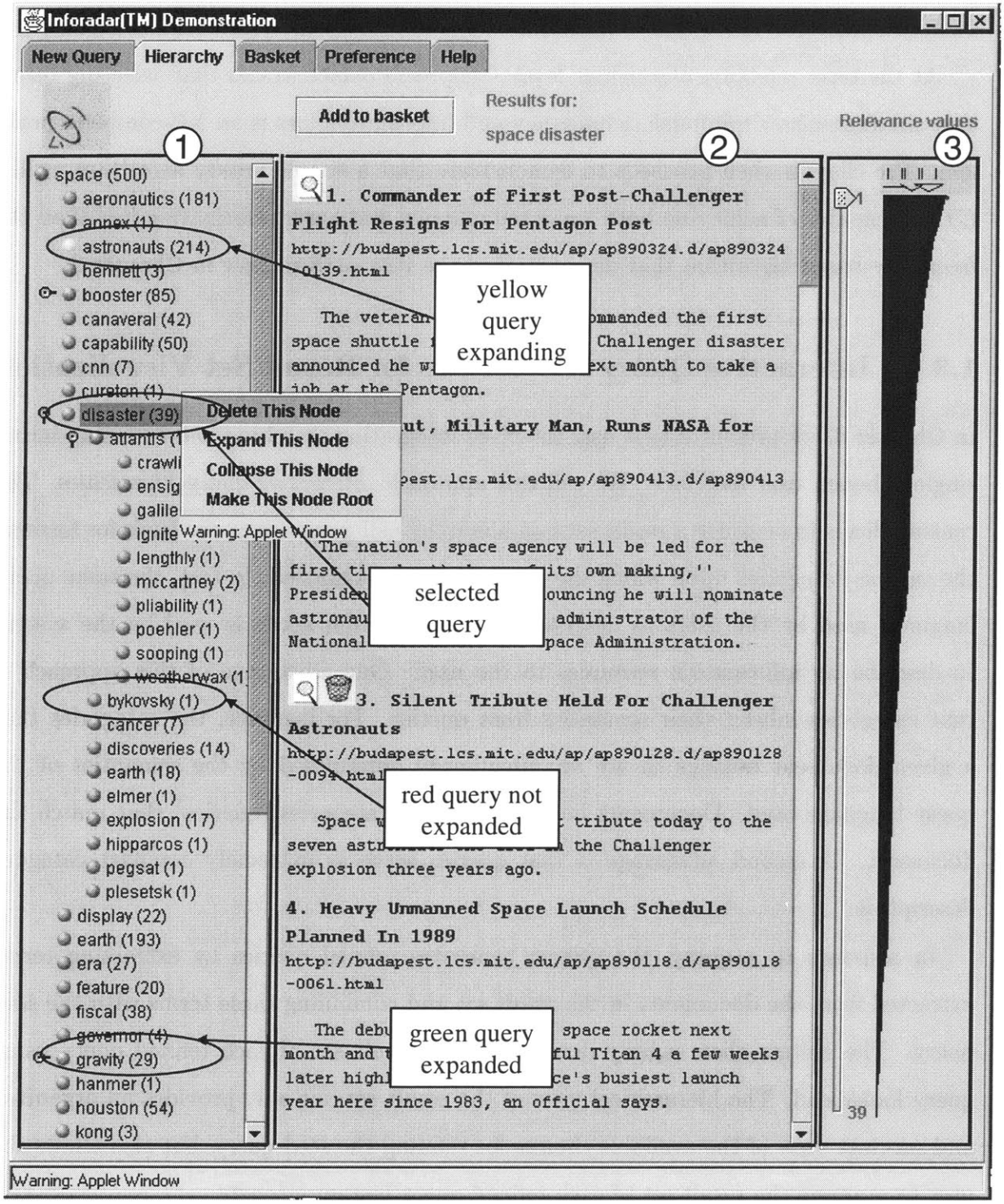



Figure 1-3: A user interface exploiting interactive query hierarchies

(www.psrg.lcs.mit.edu/projects/Inforadar) that we built as part of our research program. The interactive query hierarchy is displayed in the left panel. The seed query, the root of the hierarchy, consists of the single term `space`. The child queries combine additional terms with the seed query. For instance, the query selected in the Figure, labeled `disaster`, represents a conjunction of the terms `space` and `disaster`. The central panel in Figure 1-3 shows the result set associated with this query. The number of documents matching each query is shown in parentheses next to the query label. For instance, (`space` $\wedge$ `disaster`) matches 39 documents. Notice that the result set of each child query is a subset of the result set for the seed query. Thus, together the child queries present a hierarchically organized view of the result set for `space`.

Figure 1-3 demonstrates the ability of the prototype system to select informative categories upon which the result set for a seed query can be organized. The user can navigate the hierarchy looking for documents of interest. The hierarchy can also be expanded by double clicking on nodes. Expansion entails generating a new subtree that replaces a leaf node. The new subtree provides a finer level of description that is circumscribed to the portion of the result set pertaining to the query being expanded.

Human-generated category classifications such as that provided by the Yahoo! Internet portal (www.yahoo.com) (Figure 1-4) represent an alternative for users for which formulating a specific query is simply too difficult of impossible. Such categories suffer from two main shortcomings. First, they are expensive to create and maintain because they require a large amount of skilled human labor. Second, they are static and therefore can only help with information needs that can be translated into one of the navigational paths provided by the hierarchy. Interactive query hierarchies are dynamic and query specific. Due to the complexity of creating and maintaining classifications by hand, human-generated category hierarchies are often static and cannot adapt to user queries. It is often not possible, for instance, to obtained a hierarchical organization of the set of documents that match a particular user query.

Interactive query hierarchies represent a new way to generate dynamic categorizations

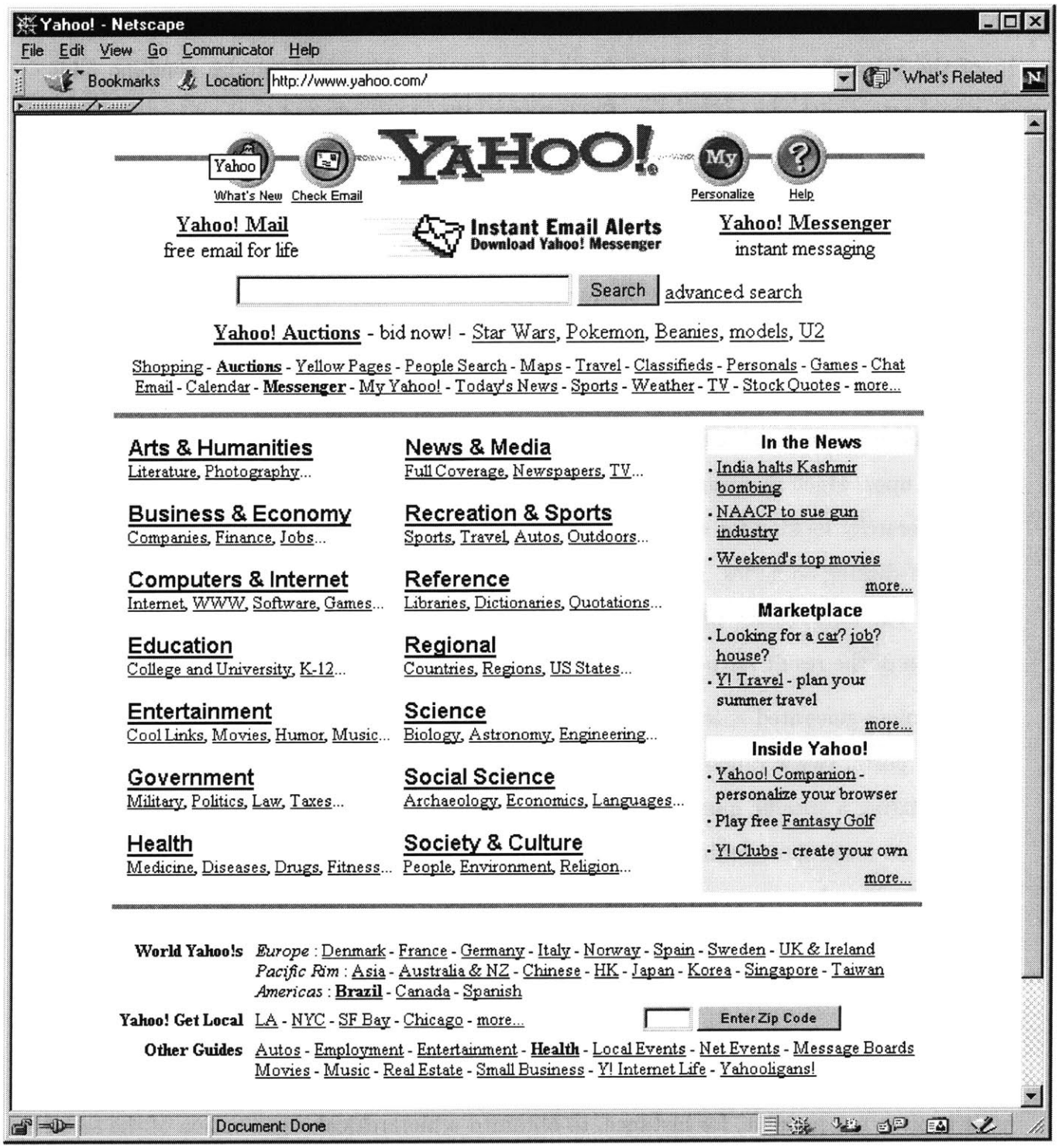



Figure 1-4: A snapshot of the initial page presented by the Yahoo! Internet portal

of arbitrary result sets without extensive human intervention. Interactive query hierarchies present the available information resources using the same query language used by users to describe their information needs. The hierarchical organization of the categories provide a context for the seed query and also present a visual description of the result set. Interactive query hierarchies can be used to either expand or refine a particular query while providing constant feedback and interaction to aid the user through the search process.

In summary, interactive query hierarchies offer the following advantages over alternative approaches to system-assisted result set visualization like document clustering and human-generated category hierarchies:

- Interactive query hierarchies avoid the semantic ambiguities inherent to document clustering by using a query language to unambiguously determine the groups of documents belonging to each category.

- A query language provides for very compact descriptions of category contents

- Interactive query hierarchies are dynamic and query specific.

As a vehicle for demonstrating that practical network search systems based on interactive query hierarchies are feasible and that such systems have the potential to improve information retrieval effectiveness, we developed the *Inforadar* prototype network search engine. Chapter 4 will provide a complete descriptions of the main components of the *Inforadar* system: the client applet, the search and refinement engine, the network protocol and the indexing module.

The *Inforadar* client applet is persistent which means that it maintains state about all previous results of user interface operations from the duration of the client session. The client session lasts from the time the applet is first loaded by the browser to the time when the applet is terminated. The document basket mechanism (Section 4.3), similar to the shopping basket found in many online shopping sites, is an example of how the

*Inforadar* client exploits persistence. The basket may hold documents found during any search operation conducted during the session.

The *Inforadar* applet can process multiple user interface operations concurrently. This means that, for instance, while the user waits for an hierarchy expansion request to be processed he/she may continue to browse the query hierarchy or perform other user interface operations. This has the effect of hiding the latency of request processing. Hiding this latency is of particular importance to us, because our search engine must spend a substantially larger amount of time than the typical search engine in order to compute the query hierarchies.

When it first receives a seed query, *Inforadar* returns, by default, a single-level query hierarchy. A user can navigate and expand the hierarchy at will according to her/his interests. Users can examine the result sets associated with child queries without having to submit the child queries to the server. These result sets are eagerly computed by the query lookahead algorithm. Hierarchy expansion, however, requires communication with the search engine. In response to an expand operation, the client applet sends the query being expanded as well as its result set to the search engine. This is necessary to ensure that the semantics of the hierarchy remain consistent across expansions.

## 1.4 Previous Work

We begin this section with a brief introduction to information retrieval system design and implementation. This introduction will facilitate the understanding of the subsequent sections discussing related research preceding this dissertation. Previous related research will be organized in three categories: system-assisted query formulation, query refinement evaluation, and system-assisted result set visualization. The degree of relatedness of previous approaches will determine the amount of effort that we will spend discussing them. For instance, the discussion of previous approaches to system-assisted query formulation will focus on previous work on query refinement. The discussion of

previous work on system-assisted result set visualization will concentrate on document clustering systems. Like interactive query hierarchies, document clustering systems attempt to group document into categories with minimal human intervention.

### 1.4.1 Overview of Information Retrieval Systems

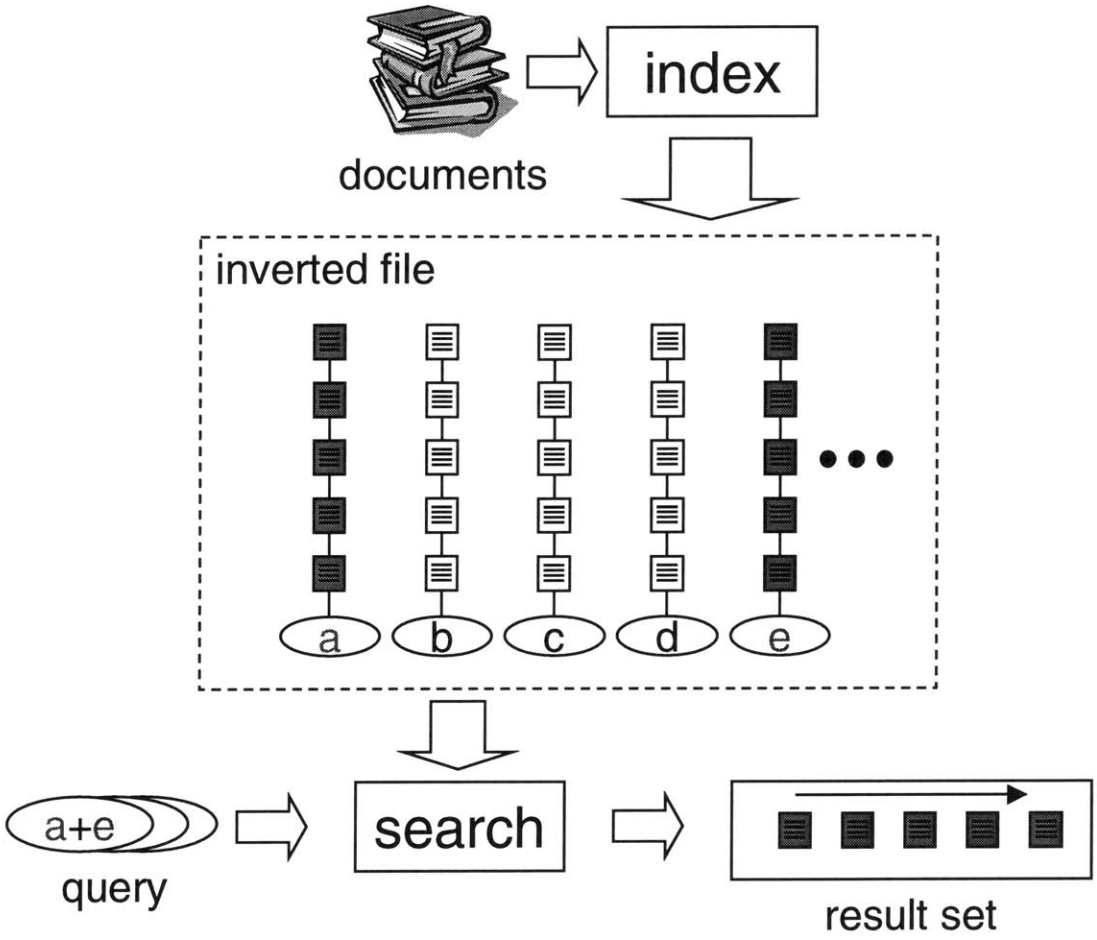



Figure 1-5: Block diagram of a generic term-based IR system

A generic full-text term-based information retrieval search system is shown in Figure 1-5. The quantum upon which such a system is based is the *search term*, or *term* for short, represented in the diagram with an oval. The search term is the finest-grain element used to describe each information object or *document* represented in the database. Information

needs are expressed in the form of *queries* which are combinations of search terms. Some concrete examples of practical search terms include words and phrases appearing in documents. Search terms may also represent metadata about documents like author or publisher information.

Information retrieval systems differ in the syntax and semantics of their query language and the mechanism they use for comparing queries to documents. In essence, however, most IR systems work in two phases that we call *indexing* and *retrieval*. The indexing phase often operates off-line and its job is to analyze or parse documents, extract their search terms, and generate a data structure, known as an inverted file. An inverted file maps each search term ever extracted to the set of documents where the term appears. The indexing phase is typically triggered when the contents of the database change.

The retrieval phase is inherently interactive in nature and operates as a read-eval-print loop that repetitively reads a user query, looks up the matching documents in the inverted file and displays the subset of available information that appears most relevant to the query. In Figure 1-5 the query combines the terms $a$ and $e$. The search module consults the inverted file data structure to find the set of documents containing each term. In this case the two shaded inverted file entries are read, corresponding to the two terms in the query. The search module proceeds by combining these two inverted file entries into a single set of documents called the *result set*. Finally the search module ranks documents according to an approximation of their relevance to the seed query.

Several query models have been proposed in the past. The most common models are the Boolean model, the vector model, and the extended Boolean model. Boolean queries combine terms using logical operators. For instance, the query `(space and shuttle)` will retrieve documents containing both terms `space` and `shuttle`. The Boolean model lacks a natural means of ranking documents in the result set as documents either match or do not match a query. The vector model represents both queries and documents as vectors consisting of terms and term weights. For instance, the vector query `(president:2 clinton:1)` will match any document containing either `president` or `clinton`, but

will give precedence to documents that contain `president`. Vector queries provide for alternative means of ranking documents according to some heuristic approximation of their relevance to the seed query. The extended boolean model provides a mathematically sound approach supporting document ranking for boolean queries. Extended boolean queries combine terms with logical operators just like boolean queries, but allow the specification of weights for any subquery like vector queries. An extended Boolean query may look like `(space:2` $\vee$ `shuttle:1)` where the term `space` has been assigned twice the weight of `shuttle`. As a result a document that contains `space` only should be ranked higher than a document containing `shuttle` only, even though both documents match the query

Following the blueprint of an information retrieval system outlined above, many different systems have been proposed, implemented and deployed in the past. At the time of this writing, such systems appear virtually everywhere that text is stored in digital form. Even after all this progress, existing information retrieval systems remain limited by the following factors:

- Information is created by humans using human languages.
- Information databases are getting larger and larger.

People, especially non expert users, like to express themselves in a human language such as English. Present state-of-the-art systems can understand many simple human language constructs, but are not fully proficient in the subtleties of human languages. Thus IR systems presently incorporating state-of-the-art human language processing technology are not clearly superior to much simpler systems relying exclusively on statistical information about term occurrence, often called *term-based systems* [53]. At the present time, minor improvements obtained by human-language-based query languages usually do not warrant the considerably larger effort needed to develop and maintain them.

It is difficult to express an information need using a term-based query. Experimental observations show that the likelihood that an information consumer and the author of

a relevant document will agree on their choice of words is small [22]. Several different terms may have equal or similar meanings (synonyms). A query containing a synonym may fail to retrieve relevant documents that deal with the same subject expressed in different terms. Alternatively, the same term may have several meanings (homonyms). A query containing an homonym may retrieve irrelevant documents that use the term but not with the meaning intended by the author of the query. Finally, many concepts cannot be expressed as a simple set of terms.

The second factor limiting the success of IR systems is the rapid pace at which the size of the underlying databases is increasing. When database sizes increase, statistical term-based systems decrease in efficiency. In a large database, even seemingly uncommon terms occur in many documents making it harder to predict which terms appear in and only in the documents of interest. In addition, query evaluation demands computational resources that are proportional to the size of the document collection. To minimize computational "waste" it is important to make the most of each query submitted to the search system.

In summary, IR systems are confronted with difficult fundamental problems. Writing precise term-based queries requires that a user predict the terms that will appear in relevant documents. As the size of the collection increases, a user's ability to make such predictions decreases. In practice it has been observed that many users end up submitting short and under-specified queries that retrieve large and imprecise result sets [13, 37, 41]. Without adequate tools, manually filtering out the noise from these results sets is an arduous task.

In response to the challenges outlined above, several solutions have been proposed. Of particular relevance to the work presented in this dissertation are previous approaches to *system-assisted query formulation*, *query refinement evaluation*, and *system-assisted result set visualization*. We will discuss previous work in each of these three areas in turn.

### 1.4.2 System-assisted Query Formulation

For gigantic collections, like the World Wide Web, a basic search system will become useless unless measures are taken to help users formulate precise queries. Several independent solutions have been proposed to the query formulation problem: *automatic query expansion*, *relevance feedback*, and *query refinement*. We will describe each of these techniques in turn.

Automatic query expansion [8, 20, 47, 48] automatically augments a *seed query* with additional terms in an attempt to increase the number of relevant documents retrieved. Sources of augmentation information include human or machine generated classifications (e.g. Dewey's decimal classification system), thesauri [52, 21], and dictionaries. Automatic query expansion has also being done using local document analysis (i.e. result sets) [69]. If a user submits `rocket` as the seed query an automatic query expander may look up the word in an Internet dictionary like the WWWebster (ww.m-w.com/cgi-bin/dictionary), extract the terms appearing in its dictionary entry, and add these terms to the user query. Some of these terms include: `firework`, `combustion`, and `propelled`. In a Boolean search system, the actual query submitted to the search module may disjunctively combine the terms from the dictionary with the seed query yielding the query `(rocket` $\vee$ `firework` $\vee$ `combustion` $\vee$ `propelled)`.

Query expansion requires no human intervention other than submission of the seed query. The lack of human intervention is also a shortcoming. Very few real systems perform automatic query expansion because adding terms to an under-specified seed query can cause the final query to diverge from the intended information need. In the previous example, terms like `combustion` and `propelled` may easily make the seed query diverge from its original intended meaning of rockets.

*Relevance feedback* [49, 31, 20] is similar to query expansion in that the goal is to expand a seed query with additional terms. The difference lies in the amount of human intervention involved and the source of the additional terms. Relevance feedback operates by requesting from a user a list of relevant documents found in the *seed result set*, the

collection of documents matching the seed query. The system then extracts terms from the relevant documents and automatically expands the query with those terms. This generates a new, hopefully more precise, result set that can be further examined and refined with further user feedback. It has been demonstrated in a number of laboratory experiments that relevance feedback can improve retrieval effectiveness [48]. However, it appears from reports by the commercial search engine community [43] that the technique has not received wide acceptance by Internet users. We believe there are two main reasons for the lack of acceptance by users. First, finding a set of relevant documents may not be a simple task. Second, once a set of relevant documents is identified, there is no user control over which terms are used to expand the seed query. Thus relevance feedback may suffer from the same divergence problems suffered by automatic query expansion.

A group at IBM has also focused on term suggestions based on relevance feedback [5]. This system allows users to annotate result documents as relevant or 'trash' and then tries to form a Boolean query that distinguishes the two sets. The selection algorithm uses a unique scoring method for determining how much terms improve a query. This scheme also requires the user to look through the result set and find positive and negative example documents.

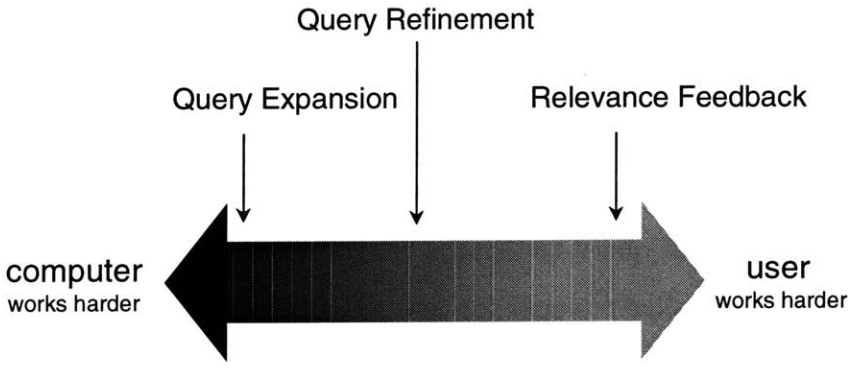



Figure 1-6: Query refinement fills a gap in the amount of human intervention required between automatic query expansion and relevance feedback

Relevance feedback relies on the user's ability and predisposition to find relevant

documents in a potentially imprecise result set. The user judges which documents are relevant while the system judges which terms should be added to the query. Query expansion also judges which terms to add to the query, but allows no human intervention. As shown in Figure 1-6 these two techniques represent two ends of a spectrum. One extreme, query expansion, requires no human intervention, placing most of the burden on the system, while the other extreme, relevance feedback, requires considerable human effort.

The third and last approach to system-assisted query formulation that we will discuss is called *query refinement*. Query refinement takes a position between the two levels of automation represented by query expansion and relevance feedback (see Figure 1-6). Query refinement automatically (and quickly) peruses the documents in a result set and selects terms to be added to the seed query. However, the term suggestions are presented to the user, who manually selects which terms to employ. This allows the user to refine the information need, to learn about the data in the collection by browsing the list of suggestions, to decide what new terms are consistent with the actual information need, and even decide which terms should expand or narrow the seed query. Term suggestions are more compact and easier to process by a human than entire documents.

Our preference for query refinement over automatic query expansion and relevance feedback is based on past experience developing the Community Information System [27, 25] as well as the Discover [58, 19, 57, 60] and HyPursuit [68] network search engines. Query refinement was originally introduced by Harman [30] although the author called the technique *interactive query expansion*. The sources for extracting term suggestions included relevant documents or similar documents as well as variants of seed query terms. Query refinement was subsequently adapted for network search engines in Discover and HyPursuit. Smeaton and van Rijsbergen [62] also use relevant documents and similar documents (nearest neighbors) to generate term suggestions.

The Open Information Locator project [36] has also investigated query refinement. They make use of two techniques: non-monotonic reasoning which combines user-directed

relevance feedback and term co-occurrence analysis, and query by navigation which allows the user to browse a hierarchy of more general and more specific terms. Unlike thesauri, co-occurrence analysis produces related terms directly from the result set of the seed query. But it cannot place them within a hierarchy. Consequently, co-occurrence lists are best used for expanding a result set rather than categorizing and refining it. This shortcoming of co-occurrence lists has also been recognized by Schatz [55], which combines co-occurrence lists with an additional hierarchical term list.

The Lexical Navigation System [11] offers query refinement based on a context thesaurus statically computed from the entire collection. The system provides both textual and graphical user interfaces for navigating a network of single and multi-word term suggestions linked by relationships automatically extracted from the corpus. A user can arbitrarily refine the seed query with additional terms from the thesaurus and submit the new query to sequentially inspect its result set.

### 1.4.3 Query Refinement Evaluation

Recall and precision are standard evaluation metrics in information retrieval [52]. Typically, measurement of precision and recall requires relevance judgments pairing each of a collection of test queries with the documents in a test corpus that are relevant to that query. Given this information, the recall and precision achieved by a set of retrieved documents can be computed by counting the number of relevant documents retrieved. Recall is the ratio of relevant documents retrieved to total relevant documents in the corpus. Precision is the ratio of relevant documents retrieved to total documents retrieved.

Recall and precision have been used in evaluating relevance feedback [32] and thesaurus query expansion techniques [69]. Aalbersberg [1] proposes a new user interface for incremental relevance feedback and measures the effectiveness of competing techniques in several standard collections based on precision metrics. Qiu and Frei [47] measure recall-precision and the usefulness of query expansions based on a similarity thesaurus constructed from the corpus. This work uses fully automatic query expansion. Efthimi-

adis [20] describes a user study in which six algorithms were used to generate query expansion terms. Users evaluated which terms might be useful for augmenting the query. Harman [30] evaluates the effectiveness of different algorithms in suggesting terms by expanding the original queries with the top 20 suggestions at the same time, and then measuring the resulting improvement in precision. This evaluation mechanism is inadequate for evaluating the impact of query refinement on short seed queries because adding many terms to a short query can easily make it diverge from its original intended meaning. Short queries are the most common queries typed by users, and thus the focus or our experiments.

Although precision improvement is an important goal for query refinement term selection, it should not be the only goal. The reason is that different information needs may be expressed with the same or very similar queries. This is particularly true for short and under-specified queries. A user who types `video` may be really looking for `video rental` or alternatively for `video compression`. A term selection algorithm that always achieves high precision improvement demonstrates that it often predicts the single information need captured by the relevance judgments used to measure precision. This may be happening at the expense of not offering the user other possible avenues for refining the query. Our concept recall metric measures another aspect of term selection effectiveness, namely semantic relatedness. The observation that precision improvement is not the only desirable property of a good set of term suggestions was one of our primary motivations for the development of the coverage-based term selection algorithms introduced in Chapter 3.

### 1.4.4 System-assisted Result Set Visualization

In addition to helping users formulate more precise queries, a system may also help them visualize result sets from naive queries. Naive queries tend to retrieve result sets that bury relevant information in irrelevant information. The problem is not matching relevant information but rather visualizing relevant information.

Several result set visualization projects have preceded our work. We have found that most of these efforts do not preclude but rather complement our approach. One approach that has received considerable attention is the development of graphical user interfaces presenting compact and intuitive views of result sets. Envision [44] displays result sets as scatter plots using properties (e.g. author names, publication dates, author specified keywords) that are assumed to be available or that can be easily extracted from documents. Veerasamy [66] developed a user interface that displays bars proportional to the frequency of occurrence of each of a set of extracted terms on each document in the result set. Thus, for each extracted term, this system displays one bar per result set document. TileBars [34] graphically depicts the frequency of occurrence of query terms within each section of a long document, thus providing relevance information at a finer level of granularity. Cat-a-Cone [35] organizes result sets using a large human generated category hierarchy (MEDLINE) and uses animation and three dimensional graphics to display the category tree together with the corresponding results. Like Cat-A-Cone, interactive query hierarchies organize result sets into categories. However, interactive query hierarchies do not have to rely on a human-generated category hierarchy.

A second approach to result set visualization applies document clustering techniques to generate summarized views of large result sets. Document clustering works by first computing some measure of similarity between each pair of documents in the collection to be clustered (e.g. a result set). Documents that are similar are grouped into clusters. A short description of each cluster is the generated. Finally, the description of each cluster is presented to a user as a summary of the information available. Most clustering techniques allow the specification of a number of clusters that most be generated. Often, the larger the number of clusters, the finer the level of detail provided by their descriptions.

Examples of search systems incorporating document clustering techniques include SenseMaker and HyPursuit. SenseMaker [3] supports alternative methods for removing duplicates and clustering result sets, and it allows users to save their results sets for later use or refinement. HyPursuit [68] also supports result set clustering using both textual

contents and link information to compute similarities between documents in a hypertext environment. More recently Kleinberg [40] has developed new algorithms for document clustering using link information.

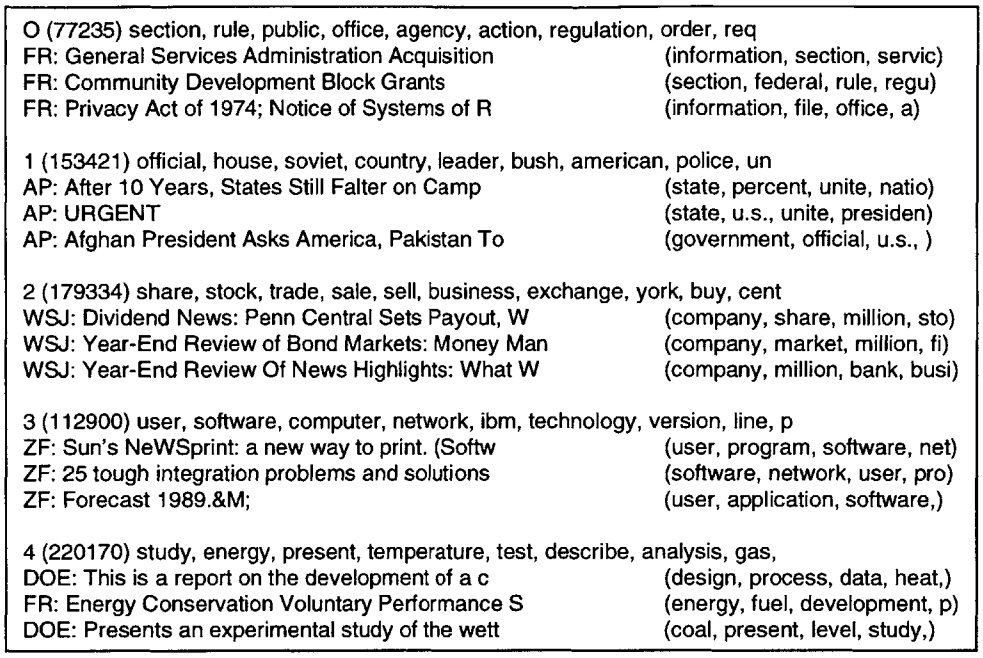



Figure 1-7: Initial screen displayed by the Scatter/Gather system. The display summarizes a large collection of documents into 5 clusters.

While the result set visualization systems discussed so far augmented search systems, Scatter/Gather [15] originally focused on applying document clustering to efficiently support browsing of large document collections. Browsing in Scatter/Gather is accomplished by iteratively displaying a view of a subset of the entire collection as a set of document clusters. As shown in Figure 1-7, each cluster is represented by a short textual description automatically generated from the documents in the cluster. Initially, the entire collection is displayed as a small set of clusters. Based on the cluster descriptions presented, the user can select one or more clusters of interest. The system then reclusters the documents in the selected clusters an presents another clustered summary, although this time at a finer level of detail. Clustering is a computationally expensive operation. In [14] it was

demonstrated that clustering can be done in constant interaction time if a data structure called a cluster hierarchy is generated by an offline process. Later, Silverstein [61] showed that clustering arbitrary result sets is possible using the same data structure in almost constant time.

Although it is clear that, under the right conditions, document clustering systems can prove to be quite useful, they suffer from a number of shortcomings limiting the scope of their usefulness:

- Documents are typically associated with only one cluster, although several cluster descriptions may be relevant to the same document
- It is difficult to generate good and compact descriptions of clusters
- Documents do not necessarily end up in the cluster whose description seems most relevant

Some of these problems are not inherent to clustering and could be fixed within the clustering paradigm. For instance, there has been recent work on "soft clustering" [45], a clustering approach that generates clusters that are not necessarily disjoint. However, we know of no clustering technology that completely solves the other two problems. Interactive query hierarchies take an alternative approach that avoids all of the above problems with clustering. As will be demonstrated in Chapter 4, using a query language to denote classification categories facilitates the generation of compact and unambiguous category descriptors.

## 1.5 Summary of Contributions

In summary, this dissertation makes the following contributions:

- Develops new experimental metrics for measuring the effectiveness of term selection algorithms for query refinement

- Invents *query lookahead*, the process of eagerly evaluating multiple refined queries derived from an initial seed query.
- Develops *LOOK*, a fast query lookahead algorithm that demonstrates that query lookahead can be integrated with query refinement at a small cost
- Invents coverage term selection and demonstrates an algorithm capable of selecting term suggestions achieving less information loss without increasing redundancy.
- Invents *interactive query hierarchies*, a new type of user interface that exploits query lookahead to provide automatic hierarchical query-based descriptions of corpus content
- Using the *Inforadar* prototype, demonstrates the feasibility of a practical search system supporting a graphical user interface supporting interactive query hierarchies

## 1.6 Structure of the Dissertation

The remainder of this dissertation is organized as follows:

Chapter 2 discusses new experimental metrics for evaluating query refinement effectiveness and presents *RMAP*, a new and faster algorithm for query refinement.

Chapter 3 introduces the idea of query lookahead and demonstrates how it can be applied to improve term selection.

Chapter 4 demonstrates how query lookahead can be exploited to devise new user interfaces providing better result set visualization tools.

Chapter 5 summarizes the most important results of this work and proposes areas where further exploration is necessary.

# Chapter 2

# Frequency-based Query Refinement

## 2.1 Introduction

Formulating precise and effective information retrieval queries has always been a difficult task, even for experienced users. Several factors contribute to this problem. The task of formulating an effective query requires the user to predict which terms appear in documents that are relevant to the information need. In addition, users want to avoid retrieving irrelevant documents due to a query that is under-specified or contains ambiguous terms. As a result, users of an information retrieval system with a large corpus are often faced with the task of manually sifting through very large and often inappropriate result sets. For example, Internet search engines such as AltaVista often return tens of thousands of hits for a query.

Query refinement is the incremental process of transforming a seed query into a new query that more accurately reflects a user's information need. Figure 2-1 illustrates the general query refinement paradigm. The paradigm involves an interactive sequence of query transformations that require minimal user effort. Users searching very large information systems, such as those indexing the World Wide Web, often start with naive queries that return too many documents to browse exhaustively. It is therefore essential that these systems provide tools for helping users focus their queries. A query refinement

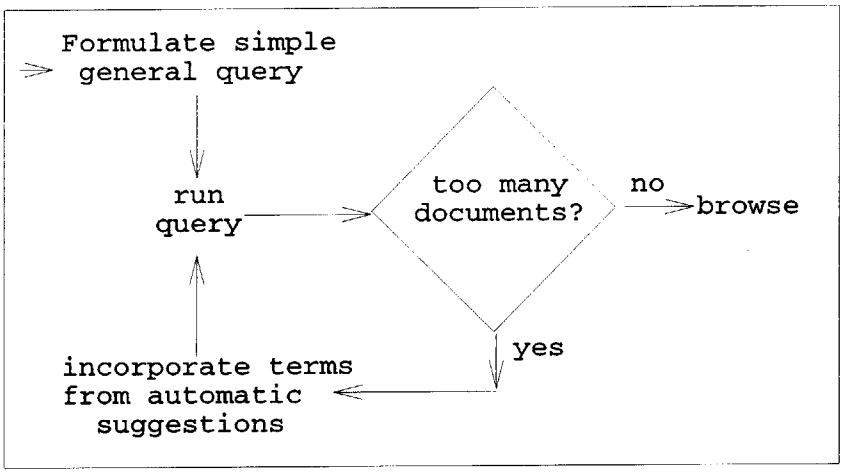



Figure 2-1: The query refinement paradigm

facility recommends automatically generated term suggestions that the user can employ as a guide for improving the seed query. The user can add suggested terms to the seed query conjunctively to reduce the result set size, disjunctively to broaden the scope of the query, by negation to focus the query away from a particular topic, or in lieu of existing query terms to modify the query. Figure 2-2 shows refinement suggestions for the seed query `text:scheme` (a programming language developed at MIT) generated by the HyPursuit prototype search tool[68]. As in the Semantic File System [26], HyPursuit allows attributes to be specified with terms to represent the portion of the document where the term should appear. In the example in Figure 2-2 attribute `text` indicates that the term `scheme` may appear anywhere inside a matching document.

At the term selection criteria level, this chapter introduces two measures of the quality of term suggestions selected by a query refinement algorithm. *Concept recall* is an experimental measure of an algorithm's ability to suggest terms that are semantically related to the user's information need. The experimental results reported in this study are based on the proportion of human selected keywords appearing in TREC (trec.nist.gov) topics [33] that were suggested by the algorithms under investigation. An algorithm that automatically generates many of these concepts is more likely to be accepted by users

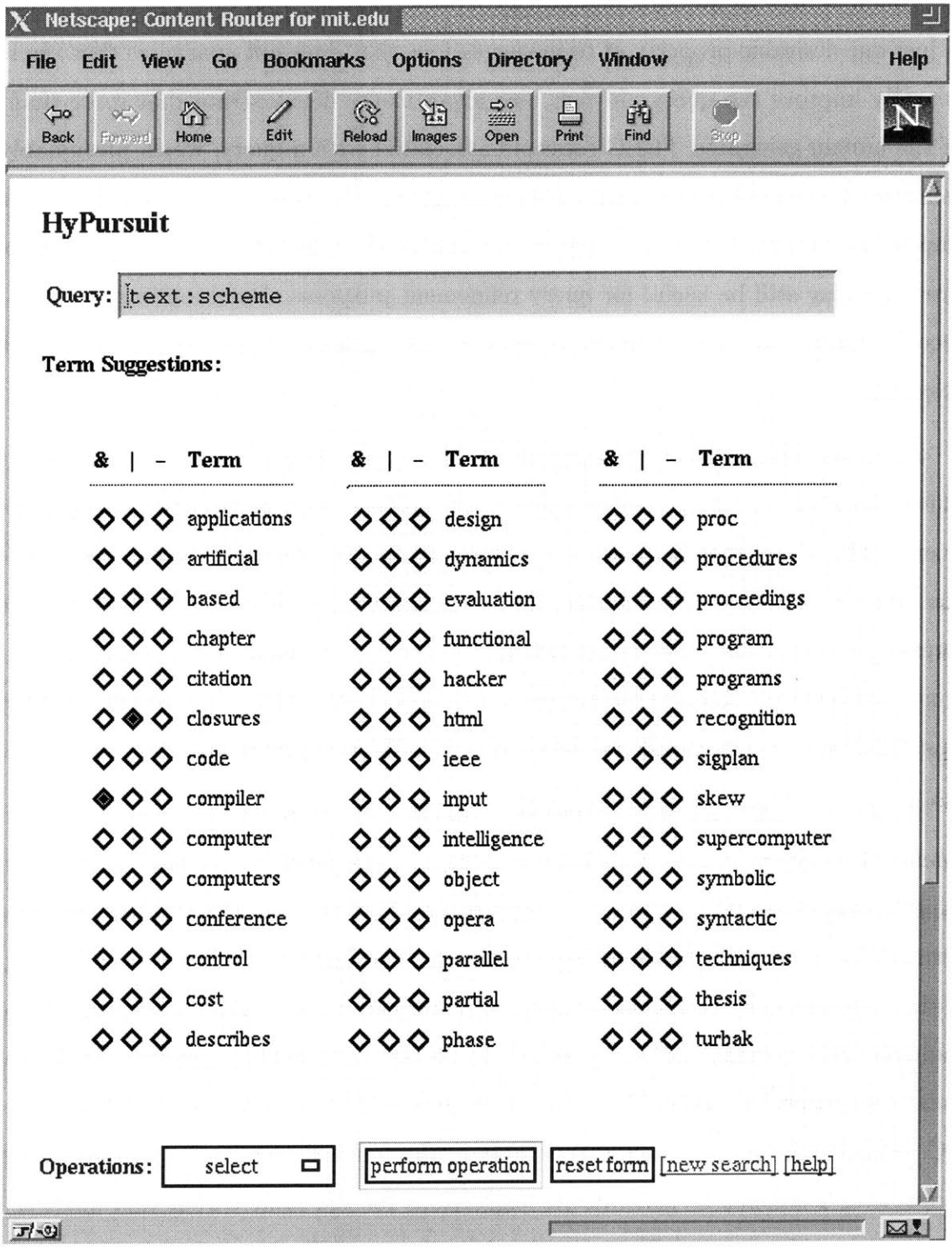



Figure 2-2: Query refinement output generated by the HyPursuit network search engine in response to the seed query `text:scheme`

because it is able to produce semantically related terms. However, semantic relatedness is just one desirable property of term suggestions that does not guarantee that the terms actually improve the query. In fact, our experiments demonstrate that more than half of the human-generated TREC concepts associated with a query, which presumably are semantically related to the query, reduce precision. However, concept recall remains an important measure because, as will be demonstrated in Section 2.4.4, terms that reduce precision may still be useful for query refinement purposes. Precision improvement, our second effectiveness metric, aims at more directly measuring this other aspect of term selection.

*Precision improvement* measures the ability of an algorithm to produce single term query modifications that predict a user's information need as partially encoded by the query. The algorithms focus on single term query modifications in order to minimize the user's effort. Our data suggests that even short query transformations are useful in improving both naive, under-specified queries as well as more advanced queries. As a control in our experimental studies we use an *oracle* algorithm. The oracle produces the best single term query modifications given a set of human generated relevance judgments.

At the term selection algorithms level, this chapter compares the accuracy and performance of two term selection algorithms using concept recall and precision improvement experiments. The DM (Document Merging) algorithm dynamically extracts and suggests terms from documents in the query result set. The RMAP (Refinement MAPping) algorithm dynamically combines precomputed suggestions for single-term queries to refine multiple term queries. RMAP is shown to be fast and effective, generating 100 refinement suggestions in under 15 *ms* in a collection of 164,000 documents using a low-end PC. RMAP achieves accuracy comparable to the much slower DM algorithm, which requires over 2 seconds to generate its suggestions for the same corpus and queries. The performance of a query refinement algorithm is especially important for low-precision, under-specified queries. For queries with an initial precision of 0–20%, the percentage of DM term suggestions that enhance precision is 25, while for RMAP it is 22. For the

same queries, the precision-enhancing terms suggested by DM improve precision by 61% of the best possible average improvement for the same number of terms, and for RMAP's terms it is 58%. Both algorithms fall short of the best possible term suggestions offered by the oracle: DM is able to suggest 43% of all terms that improve precision by 10% or more, while RMAP suggests 37% of these terms.

In the remainder of this chapter we present the two query refinement algorithms that are the object of this study (Section 2.3), describe our experimental results (Section 2.4) and offer conclusions and possible avenues for future work (Section 2.5).

## 2.2 Term Selection Criteria

This section introduces two criteria for selecting effective term suggestions for query refinement. First, term suggestions should be semantically related to the seed query. Although clearly an algorithm that selects terms that are not semantically related to the seed query cannot be expected to perform well, semantic relatedness is not the only desirable property of effective term suggestions. A second criterion is that term suggestions should improve the precision of the seed query. As will be demonstrated experimentally in Section 2.4, terms that are semantically related to the seed query may not improve precision. In fact, some of these terms may decrease precision.

Precision improvement, on the other hand, is not the only desirable property of term suggestions either. The reason is that the same query could be used to encode multiple information needs. This is particularly true about short under-specified queries, the type of query most often submitted by users. A simple example is when a user types a short query even though he/she has a more specific information need in mind. A term that increases precision does so along one and only one of the possible information needs encoded by a short query. Other terms which may reduce precision may just be focusing the query along another of the possible information needs. We will further illustrate this point in Section 2.4.

In summary, sematic relatedness and precision improvement are two complementary criteria for selecting terms for query refinement. Neither criterion can individually provide all the information necessary to measure effective term selection.

In the remainder of this section we introduce two experimental metrics, concept recall and precision improvement, that measure how well a term selection algorithm satisfies each of the above two criteria. Both metrics rely on information provided by some standard collection consisting of documents, sample queries, and human generated relevance judgments. The relevance judgments encode the information need corresponding to a query by enumerating the documents in the corpus relevant to the information need. An example of such a standard collection is the TREC collection [33] distributed by the National Institute of Standards and Technology (trec.nist.gov).

### 2.2.1 Concept Recall

*Concept recall* is a measure of the ability of an algorithm to recommend terms that are semantically related to the user's information need. The idea is to compare the terms selected by an algorithm with sets of precomputed terms known to be semantically related to the query.

Figure 2-3 shows a example TREC topic. A TREC topic includes the following fields: a numeric identifier, a domain description, a topic field, a description of the query, a narrative that further elaborates the information need, and concepts related to the query. The terms appearing in the topic field are used in our experiments as test queries. The terms appearing in the concepts field include keywords judged by humans to be semantically related to the information need of the topic. Using these terms we were able to define the concept recall achieved by a given test query as the ratio of concepts suggested by the algorithm as refinements for that query over the total number of concepts associated with the corresponding topic.

**Number** 008

**Domain** International Economics

**Topic** Economic Projections.

**Description** Document will contain quantitative projections of the future value of some economic indicator for countries other than the U.S.

**Narrative** To be relevant, a document must include a projection of the value of an economic indicator (e.g., gross national product (GNP), stock market, rate of inflation, balance of payments, currency exchange rate, stock market value, per capita income, etc.) for the coming year, for a country other than the United States.

**Concepts**

1. inflation, stagflation, indicators, index
2. signs, projection, forecast, future,
3. rise, shift, fall, growth, drop, expansion, slowdown, recovery,
4. billions,
5. Not U.S.

Figure 2-3: An Example TREC Topic. Short test queries are obtained from the Topic field. The "Concepts" field is used as human generated suggestions by the concept recall experiments.

### 2.2.2 Precision Improvement

Our second metric, *precision improvement*, measures the performance of a query refinement algorithm by examining the effect of adding suggested terms to the seed query. First, we compute the precision of the result set for each test query. Then, each algorithm produces a fixed number of term suggestions for every query. For each query, we construct a set of modified queries by adding each suggested term in turn to the seed query. Finally, we measure the precision of the result set for each of the modified queries. In computing precision, we consider only the top 100 documents matching a

query. The ranking algorithm used ranks documents with a larger proportion of query terms higher. This seemingly simple approach performed better that other approaches in our experiments.

## 2.3 Term Selection Algorithms

This section introduces two query refinement algorithms that we have implemented and evaluated. Section 2.3.1 presents a generic query refinement algorithm, DM (Document Merging), based on our prior work [59, 19, 68] that can be specialized by selecting a term weighting function. Section 2.3.2 describes and analyzes RMAP, a new and faster query refinement algorithm. Section 2.3.3 analyzes the time and space complexity of DM and RMAP.

### 2.3.1 The DM Family of Algorithms

Figure 2-4 illustrates the operational flow of the DM algorithms. Given a seed query and a corpus, DM first finds all documents matching[1] the seed query. It then combines and ranks the terms in these documents, and finally displays the highest ranked terms as query refinement suggestions. The distinguishing parameter for the DM algorithms is the weight function used to select the particular subset of terms. Thus $\mathrm{DM_{df}}$, $\mathrm{DM_{tf}}$, and $\mathrm{DM_{nfx}}$ refer to DM instantiated with the $W_{\mathrm{df}}$, $W_{\mathrm{tf}}$, and $W_{\mathrm{nfx}}$ term weighting functions respectively. These functions are described below. More precisely, for a given weight function *fcn*, $\mathrm{DM}_{fcn}$ operates according to the steps shown in Figure 2-5.

The goal of a weight function $W_{fcn}(S)$ is to approximate the effectiveness of the terms in the selected suggestion set $S$ in helping the user focus the query. This chapter studies weight functions where the weight $W_{fcn}(S)$ is the sum of the individual weights of each term in $S$. In this case, the selected set $S$ of term suggestions consists of the top $n$

[1]This chapter is purposely vague about the definition of "matching". The algorithms described are independent of the specific matching mechanism made by different query models.

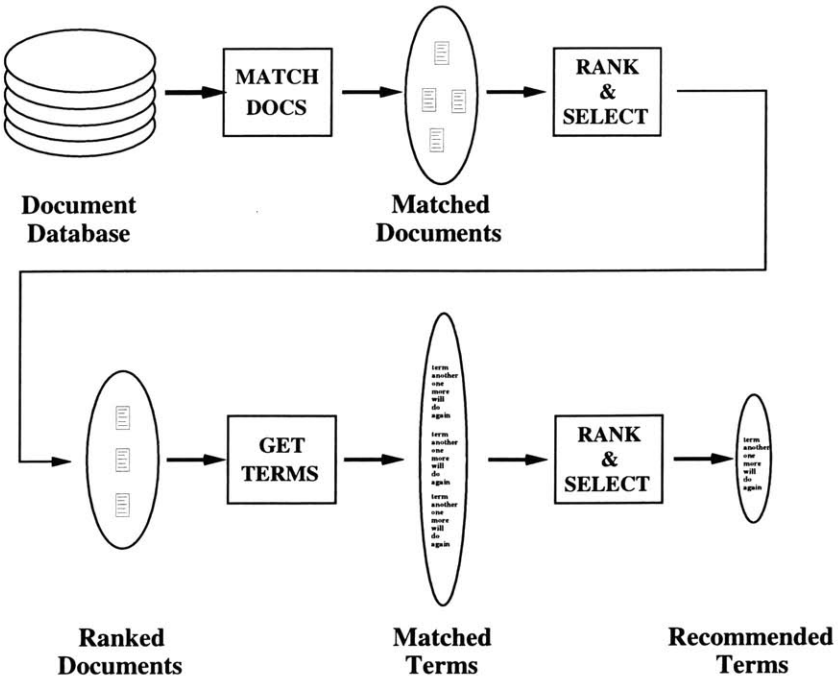



Figure 2-4: Flow diagram of the DM algorithms.

ranked terms in $\mathcal{F}(q)$. For each algorithm *fcn* below, the function $w_{fcn}(q,t)$ computes the individual weight of a term $t$ in refining query $q$. We are currently investigating other weight functions that consider the weight of a term set as a whole. For example, an alternate algorithm can weight a set of suggestions based on the number of matching documents that contain at least one of the terms from the set.

Figure 2-6 shows the three term weighting functions studied in this chapter. It is important to point out that all of our term selection algorithms use the common information retrieval technique of ignoring extremely common words appearing in a stop list. The first function, $w_{\text{df}}$ assigns to each term $t$ a weight equal to the document frequency of the term in the set of documents matching the query. The Discover system [19] uses $w_{\text{df}}$ and provides anecdotal evidence that this approach generates useful query suggestions based on a collection of WAIS [38] document headlines.

The second weight function, $w_{\text{tf}}$, is based on term frequencies in the matching doc-

Let

$C$ = document corpus
$q$ = seed query
$r$ = number of matching documents to scan
$n$ = number of suggestions to display
$W_{fcn}(S)$ = algorithm specific weight of term set S

1. Compute the set of documents $\mathcal{D}(q) \in C$ that match the query $q$.
2. Select a subset $D_r(q)$ of top $r$ matching documents
3. Compute the set of terms $\mathcal{F}(q)$ from the documents $D_r(q)$ such that $\mathcal{F}(q) = \{t | \exists d \in D_r(q) : t \in d\}$ where $d$ is a document and $t$ is a term.
4. Compute the subset $S$ of $n$ terms from $\mathcal{F}(q)$ with the highest weight $W_{fcn}(S)$.
5. Present $S$ to the user as the set of term suggestions.

Figure 2-5: *DM* query refinement algorithm

uments. The third weight function, $w_{\text{nfx}}$, adjusts term frequency weights with inverse document frequencies to reduce the weights of terms that appear frequently in the corpus. $w_{\text{nfx}}$ also normalizes term weights by document sizes to counteract the increase in the weights of terms that appear in large documents.

Other term weighting functions are possible, although not studied in this chapter. For instance it is also possible to use term weighting functions that assign more weight to terms that appear in documents ranking higher with respect to the query. Chapter 3 will present yet another scheme for weighting terms called coverage term selection.

### 2.3.2 RMAP: Fast Query Refinement

RMAP achieves greater speed than the DM algorithms by statically precomputing a substantial amount of work that is performed dynamically by DM. RMAP thus uses more

| weight | function |
|---|---|
| $w_{\mathrm{df}}(q,t)$ | $\sum_{d \in (\mathcal{D}(q) \cap D(t))} \mathrm{df}_t$ |
| $w_{\mathrm{tf}}(q,t)$ | $\sum_{d \in (\mathcal{D}(q) \cap D(t))} \mathrm{tf}_{t,d}$ |
| $w_{\mathrm{nfx}}(q,t)$ | $\sum_{d \in (\mathcal{D}(q) \cap D(t))} 0.5 + \frac{0.5(\mathrm{tf}_{t,d})}{\max_{i \in d} \mathrm{tf}_{i,d}} * log\left(\frac{N}{\lvert D(t) \rvert}\right)$ |

Figure 2-6: Term weighting functions used with the DM algorithm to obtain reported experimental results. $D(q)$ stands for the set of documents matching query $q$, $N$ is the size of the corpus, $\mathrm{df}_t$ is the document frequency of term $t$, and $\mathrm{tf}_{t,d}$ is the number of occurrences of term $t$ inside document $d$.

storage space to avoid online computation. Specifically, RMAP operates in two phases: an offline corpus preprocessing phase, and a dynamic phase in response to a user request for term suggestions. During the offline phase, the algorithm generates a data structure that maps each term $t$ in the corpus to an RSET, which is the set of $m$ terms that the DM algorithm would suggest given the single term query $t$. The online computation only consists of term lookup and ranking. Because the RMAP data structure can be updated incrementally in response to changes in the composition of the corpus, there is no need to rerun the entire offline process when changes occur.

The hypothesis underlying RMAP is that relatively small values of $m$ will not result in significant decreases in accuracy versus the purely dynamic algorithms. Preliminary results suggest that reducing $m$ to values close to, or even below, the number of required term suggestions does not degrade the performance of RMAP significantly. Section 2.4 presents experimental and anecdotal evidence that RMAP with $m = 100$ achieves accuracy comparable to DM with $r = 100$. The length $m$ of each RSET can be adjusted to meet different storage requirements. An optimal choice of $m$ will take into consideration many factors including corpus size, disk space, performance constraints, and term distribution.

Figure 2-7 illustrates the dynamic phase of RMAP that is performed in response to a

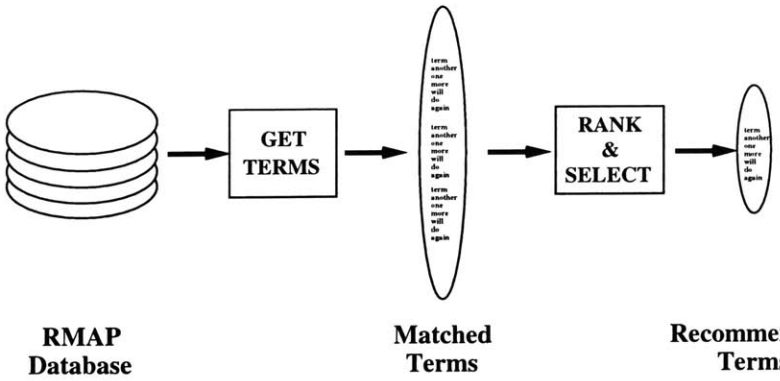



Figure 2-7: The dynamic phase of the RMAP query refinement algorithm.

request for query refinement suggestions. For each term in the seed query, the algorithm looks up the corresponding RSET containing both terms and the weights assigned to them during the static phase. All the terms are merged into a single set of terms. The new weight for each term is the sum of its weights from the RSETs. The highest weighted $n$ terms are displayed to the user as suggestions.

### 2.3.3 Time/Space Complexity

This section compares the time and space complexity of RMAP and DM. The average running time for DM is the sum of the average running times of its four phases. We will analyze the time complexity of both algorithms using the size of the result set and the cumulative number of terms among all result set documents as the appropriate measures of input size. Analysis of the MATCH DOCS phase would require information on the frequency distribution of queries and their result sets which is not available to us.

Let

$rss$ = size of the result set

$r$ = number of matching documents to consider

$cds$ = cumulative number of terms among all result set documents

$n$ = number of suggestions to display

- RANK & SELECT takes on average $rss + r$
- GET TERMS takes on average $cds$
- RANK & SELECT takes on average $cds + n$

The average running time for RMAP is the sum of the average running times of its two online phases:

- GET TERMS takes on average $n_q \cdot m$
- RANK & SELECT takes on average $n_q \cdot m + n$

Any optimization in query processing technology that can be applied to the MATCH DOCS and GET TERMS phases of DM can also be applied to the dynamic lookup phase of RMAP. For example, a search engine may extract and rank only a portion of the results. Furthermore, a system may substantially improve running times by keeping index and RMAP data structures in main memory rather than on disk. Our implementations of RMAP and DM use exactly the same code for database lookup and term ranking/selection tasks.

The improvement in time complexity of RMAP's dynamic phase over DM comes at a price in storage requirements. Specifically, in a corpus with $T$ unique terms RMAP requires space of size $O(T \cdot m)$ in addition to the space required by DM. Both RMAP and DM require space to store an inverted file and additional storage to hold each document's terms. Section 2.4 will present space requirements for both RMAP and DM for a sample of collections of different sizes.

## 2.4 Evaluation of Algorithms

This section compares the performance of the algorithms presented in Section 2.3 using the criteria for term selection introduced in Section 2.2. The first evaluation metric,

| # of Documents | $\overline{df}$ | $\overline{dt}$ | Unique Terms |
|---|---|---|---|
| 8,466 | 28.87 | 138.76 | 40,684 |
| 84,660 | 91.31 | 141.64 | 131,314 |
| 164,597 | 180.92 | 203.98 | 185,524 |

Table 2.1: Statistics on the test collection used for the evaluation of the DM and RMAP query refinement algorithms.

| # of Queries | Avg Query Length | Relevant Docs/Query | | |
|---|---|---|---|---|
| | | 8,466 | 84,660 | 164,597 |
| 150 | 2.08 | 17.30 | 187.64 | 364.83 |
| 200 | 2.84 | 18.73 | 203.36 | 390.36 |

Table 2.2: Statistics on the test queries used for the evaluation of the DM and RMAP query refinement algorithms.

*concept recall*, measures the proportion of suggested terms that appear in a list of human generated relevant terms. This measures an algorithm's ability to suggest terms semantically related to a given query. The second approach, *precision improvement*, measures the effectiveness of individual terms in modifying an under-specified query to better approximate the user's information need. The discussion proceeds by comparing the run-time performance of the different algorithms and offering an analysis of their storage requirements. It concludes by examining sample suggestions from our prototype. The experiments were conducted using the *Inforadar* search engine described in Section 4.5. The search engine supports a Boolean query model extended with document ranking. Our ranking algorithm differs from classical ranking algorithms such as [31] by ignoring document size and by giving primary consideration to the number of query terms a document contains. The ranking algorithm is kept constant across the different query refinement algorithms evaluated in this section.

### 2.4.1 Collections and Query Sets

The experimental framework uses a standard collection that consists of documents, sample queries, and human generated relevance judgments. The relevance judgments encode the information need corresponding to a query by enumerating the documents in the corpus relevant to the information need. Tables 2.1 and 2.2 show relevant statistics about the collections used. Each experimental collection contains AP press articles published during 1988 and 1989 appearing in volumes 1 and 2 of the TREC CDROMs. The collection with 164,597 document contains all the articles published during these two years. That with 84660 documents contains all the articles published in 1989. The smallest collection contains the first 8,466 articles published in 1989.

The experiments below construct queries using the terms from the *topic* field of TREC topics (Figure 2-3). Many researchers use the description and narrative fields as the basis of the query because it more accurately reflects the information need specified in the detailed narrative. However, we chose to use the topic field because the size of the resulting queries is more representative of the size of queries made by users in large information systems. For example, Croft et al. [13] determined that searches on the World Wide Web have on average only two terms. Using the topic field results in an average query length of 2.08 for the set of 150 queries and 2.84 for the set of 200 queries. Query refinement attempts to bridge the gap between the short, under-specified queries users Initially submit and the more detailed information needs that users have.

A query can be an encoding of many different information needs. This is especially true of the short queries frequently posed by naive users, which we model using the TREC topic field. The accuracy results reported below are measured with respect to TREC relevance judgments based on specific information needs not fully encoded in our queries. If a query refinement algorithm performs well over a range of queries and information needs, then it is likely that this refinement algorithm generally suggests useful query modifications, and thus achieves its goal in practice.

### 2.4.2 Concept Recall

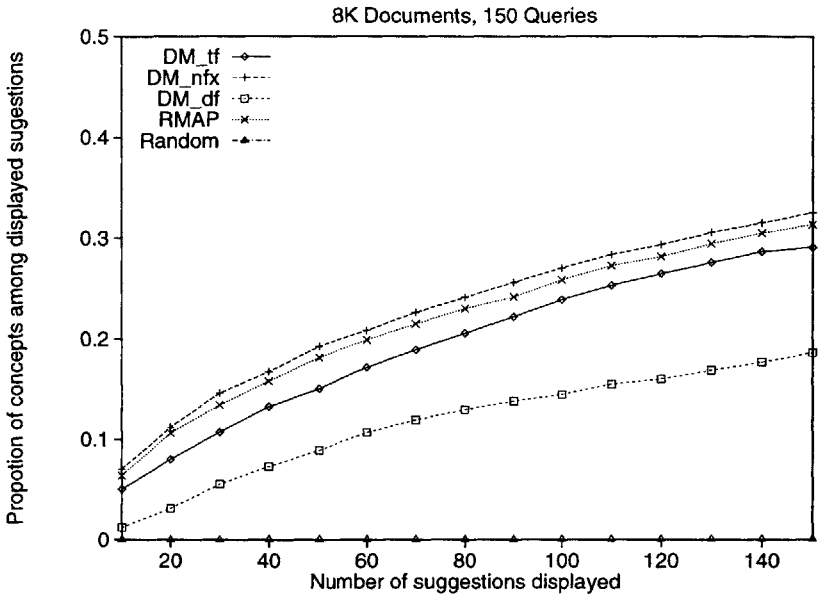



Figure 2-8: Proportion of concepts among the displayed suggestions (8K Docs collection).

Figures 2-8 and 2-9 illustrate the concept recall results for the different algorithms on corpora of 8466 and 84660 documents, respectively. The graphs show the average concept recall (over 150 queries) versus the number of term suggestions displayed. Concept recall ratios ignore concepts that do not appear in the corresponding collection. The average number of concepts per topic is 18.30, and on average 15.7 (2,350 total) of these appear in the smaller corpus while 16.3 (2,449 total) appear in the larger corpus. For example, for a corpus of 84,660 documents, the $DM_{nfx}$ algorithm was able on average to automatically suggest 35% of the human generated concepts when displaying only 100 term suggestions. The faster RMAP algorithm performs better than $DM_{tf}$ and $DM_{df}$ and does not significantly lag behind $DM_{nfx}$. The graphs also include a control that suggests random terms from the corpus. The two figures illustrate an improvement in concept recall that results from increasing the size of the corpus from 8,466 documents to 84,660 documents and confirm the well known positive impact of increasing collection size on

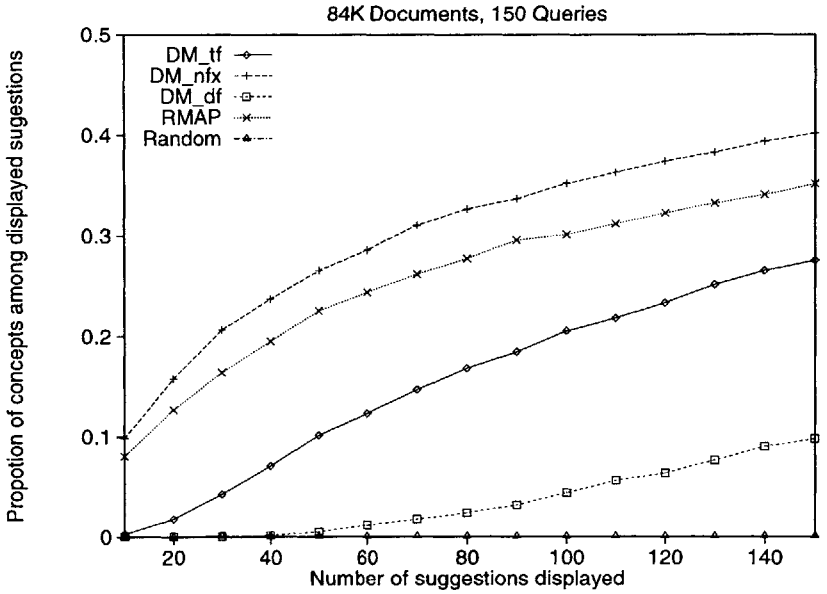



Figure 2-9: Proportion of concepts among displayed suggestions (84K Docs collection).

the utility of inverse document frequency (*idf*) factors.

### 2.4.3 Precision Improvement

This section presents several views of precision improvement results for the $DM_{nfx}$ and RMAP algorithms. It does not present results for $DM_{df}$ and $DM_{tf}$ because these algorithm demonstrated lower performance than $DM_{nfx}$ and the same running times.

Figure 2-10 shows the distribution of the 200 test queries examined in the precision improvement experiment. The initial precision measures the precision of the query before being modified with additional terms. More than 50% of the test queries have 50% precision or less.

The performance data for an algorithm underestimates the actual utility of its suggested terms because a suggestion may be useful even if it does not improve precision. For example, for the TREC topic `economic projections` (presented in Section 2.4.1), the query suggestions `budget`, `deficit`, `loan`, and `debt` all decrease precision. Yet these

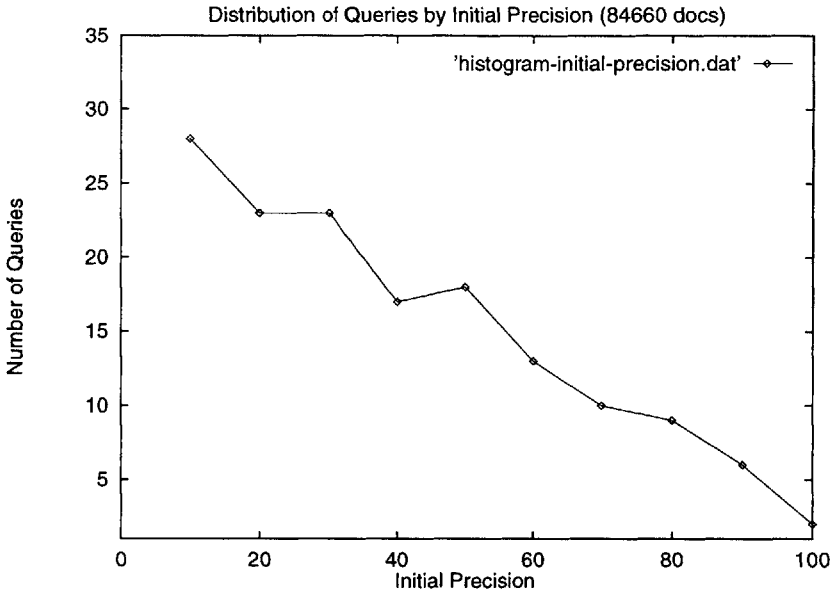



Figure 2-10: Distribution of test queries according to their initial precision

terms are clearly related to the query, and moreover would be useful for information needs other than the one specified in the TREC topic. See Section 2.4.4 for more example term suggestions along with a detailed discussion of this issue.

The above examples are indicative of the general pattern that semantically related terms may not necessarily improve precision. Figure 2-11 shows that more than half of the user generated concepts in the TREC topics actually reduce precision. The graph displays the distribution of changes in precision resulting from adding individual concepts to 150 test queries. The $x$ axis denotes the change in the number of relevant documents in the top 100 retrieved after adding a term to the seed query. The $y$ axis bars represent the number of terms that change the precision over the interval (exclusive of the lower $x$ value and inclusive of the higher $x$ value).

The remainder of this section presents an analysis of the distribution of the effects of single-term query modifications both in the aggregate and in relation to initial query precision.

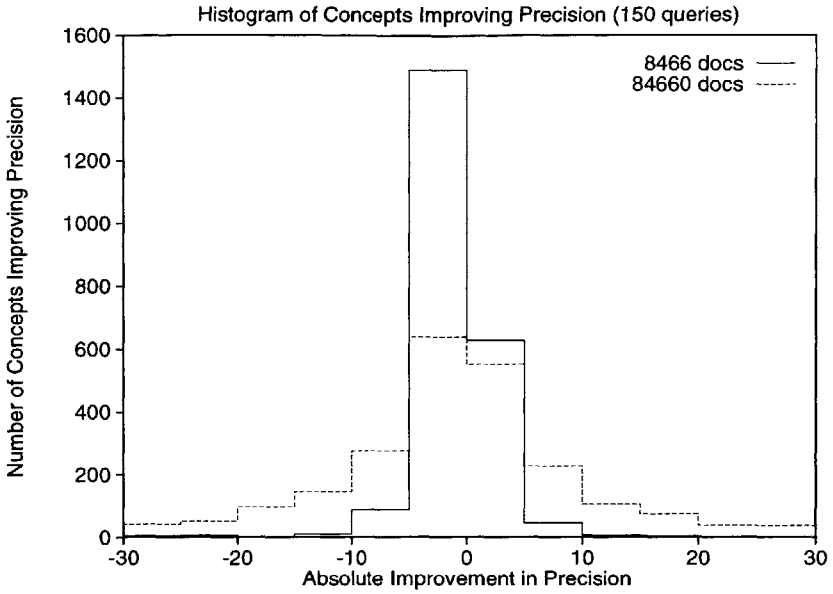



Figure 2-11: Effect of TREC topic concepts on query precision

### Distribution of Changes in Precision

The histogram in Figure 2-12 shows the absolute changes in precision resulting from modifying each of 200 queries from the standard collection. For each test query we form 100 modified queries by adding the refinement terms suggested by RMAP and $DM_{nfx}$ for that query. For example, out of 20,000 suggestions for all queries, $DM_{nfx}$ suggests 2,256 terms that improve precision by adding more than 0 documents and less than or equal to 5 documents. Though RMAP is substantially faster, it performs comparably to $DM_{nfx}$ in recommending terms that improve precision.

For comparison, Figure 2-13 illustrates the performance of two control algorithms, *Random* and *Oracle*. Random simply suggests terms from the collection at random. Intuitively, most terms in the collection are orthogonal to the query. Random shows that orthogonal terms are likely to have little effect on precision.

Oracle knows all documents relevant to each query and suggests the terms that improve precision the most. In this experiment, oracle extracted the terms from all documents relevant to a query, and measured the effect of each term on precision. It then

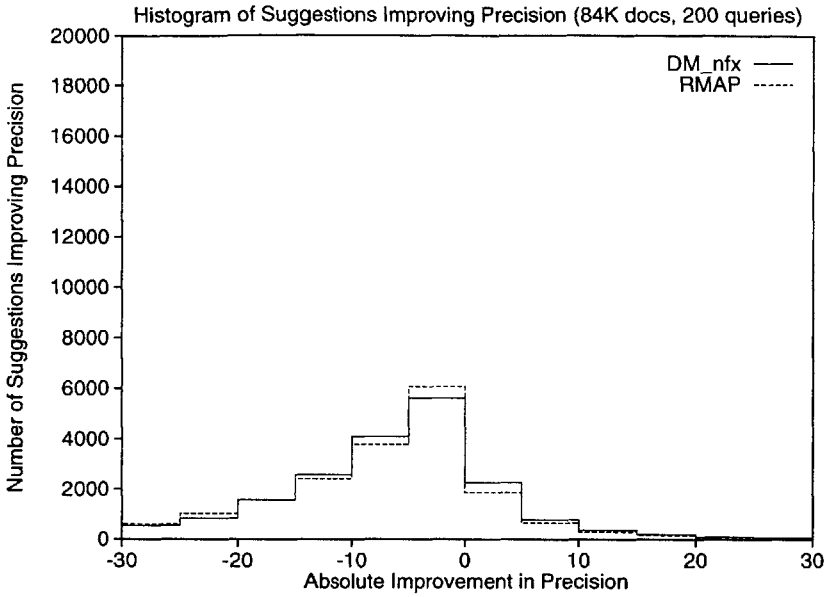



Figure 2-12: Distribution of changes in precision. DM versus RMAP algorithms.

suggests the top 100 terms. Oracle's performance is therefore the best that any algorithm can hope to attain. Note that even in oracle's case, 10% of the terms do not improve precision (these 1,962 terms have a value of 0).

Most of the terms suggested by RMAP and $DM_{nfx}$ are correlated to the query because they have a noticeable positive or negative effect on precision. Oracle shows that there are only a few terms that substantially improve precision, and RMAP and $DM_{nfx}$ suggest a significant portion of them. For example, $DM_{nfx}$ is able to suggest 43% of all terms that improve precision by 10% or more, while RMAP suggests 37% of these terms. RMAP and $DM_{nfx}$ are less able to suggest a significant proportion of the terms that improve precision modestly.

### Improvement versus Initial Query Precision

Queries with low initial precision have a lot of room to improve but do not carry much information. Conversely, queries with high initial precision will be difficult to improve

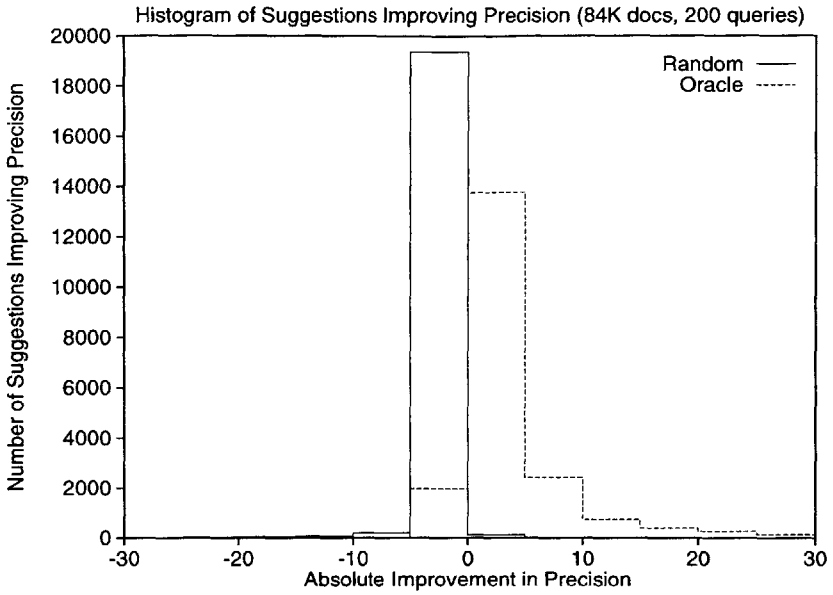



Figure 2-13: Distribution of changes in precision. Random versus Oracle algorithms

upon, but they have a high information content. This section investigates the combined effect of the two factors of information content and query specificity.

Figure 2-14 shows the percentage of suggested terms that improve precision versus the precision of the initial query. For example, $DM_{nfx}$ on average suggests 29 (out of 100) terms that improve precision for queries with an initial precision of $10 - 20\%$. For queries with an initial precision of $0 - 20\%$, the percentage of $DM_{nfx}$ term suggestions that enhance precision is 25%, while for RMAP it's 22%. The average performance of $DM_{nfx}$ over all queries is 20%; the average for RMAP is 16%. We suspect that since RMAP is a global algorithm and DM is a local algorithm (in the sense of [69]), DM is better able to take advantage of information rich queries.

Figure 2-15, which considers only suggestions that improve precision, shows the number of new relevant documents out of 100 added to the result set versus initial query precision. For example, RMAP's suggestions that improve precision result in an average of 9 additional relevant documents among the top 100 for queries with an initial precision

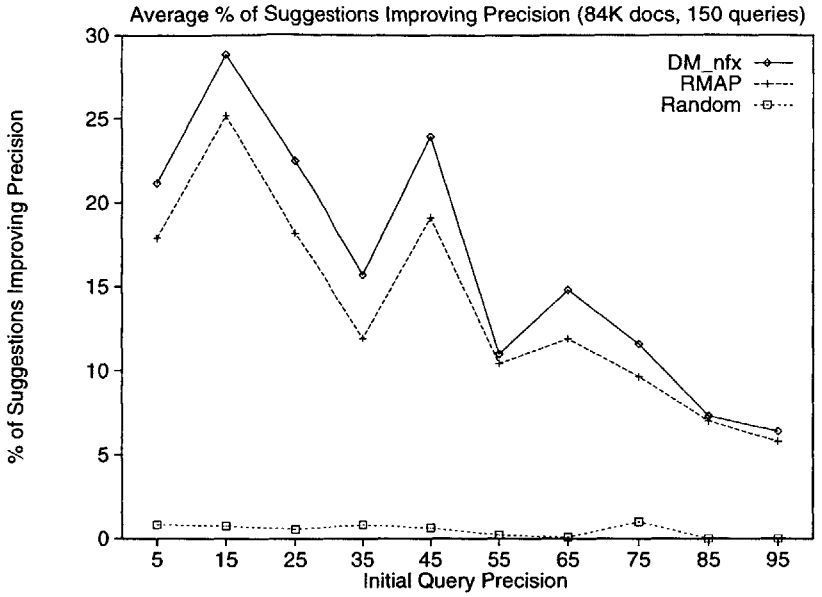



Figure 2-14: Percentage of Suggestions Improving Precision

of $10-20\%$. The average across all initial query precisions for both algorithms is 7. This figure does not show the performance of Random because the average percentage of its suggestions that improve precision is low enough to render the algorithm useless.

The oracle values depicted in Figure 2-15 represent ideal performance. The oracles for $DM_{nfx}$ and RMAP, denoted by `oracle(#`$DM_{nfx}$`)` and `oracle(#RMAP)` respectively, generate as many precision-enhancing terms as the corresponding algorithm. Thus, if RMAP recommends $n$ precision-enhancing terms for a particular query, the oracle for RMAP suggests its top $n$ terms for the same query. The performance of RMAP and $DM_{nfx}$ converges to the performance of their respective oracles as initial query precision increases. For queries with an initial precision of $0-20\%$, the average improvement in precision of terms suggested by $DM_{nfx}$ is 49% of the best possible average improvement for the same number of terms, and for RMAP's terms it is 51%. For queries with an initial precision of $70-80\%$, $DM_{nfx}$ achieves 70% of the best possible average improvement and RMAP achieves 71%. The overall average improvement for precision-enhancing terms suggested by $DM_{nfx}$ is 7%, which is 61% of the best possible for the same number of

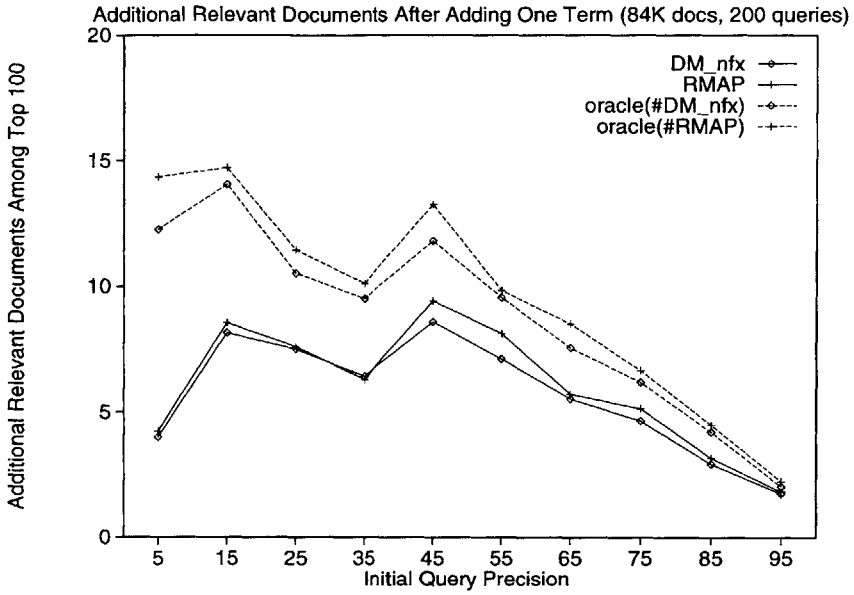



Figure 2-15: Average Improvement in Precision

terms, while the overall average for RMAP is also 7%, which is 58% of the best possible.

### 2.4.4 Example Term Suggestions

Figure 2-16 shows the actual terms suggested by the $DM_{nfx}$ and RMAP algorithms in response to the seed query `Economic Projections` extracted from TREC topic 008 shown in Figure 2-3. The query achieves an initial precision of 17%. The column labeled "Oracle Rank" contains the position of each term among the set of oracle suggestions. For each term, the columns labeled "In $DM_{nfx}$?" and "In RMAP?" indicate whether the algorithm suggests the corresponding term. The "+ Docs" column displays the total number of additional relevant documents among the top 100 ranked documents retrieved by the modified query after adding the corresponding term. The last column titled "+% Docs" displays the same information as a percentage of the number of relevant documents retrieved by the seed query. All the terms are shown stemmed. As was suggested to in the discussion regarding Figures 2-12 and 2-13, both algorithms retrieve many of the top precision improving terms but are less able to retrieve terms that only modestly improve

| Term | Oracle Rank | In $DM_{nfx}$? | In RMAP? | + Docs | + % Docs |
|---|---|---|---|---|---|
| forecast | 1 | x | | 22 | 129 |
| growth | 2 | x | x | 19 | 111 |
| percent | " | x | x | 19 | 111 |
| rise | 5 | x | | 18 | 105 |
| inflat | 7 | x | x | 15 | 88 |
| rate | " | x | x | 15 | 88 |
| slow | 10 | x | | 13 | 76 |
| economi | 13 | x | x | 11 | 64 |
| economist | " | x | | 11 | 64 |
| increas | 19 | x | | 9 | 52 |
| gross | 21 | x | | 8 | 47 |
| price | 26 | | x | 7 | 41 |
| annual | 36 | x | | 6 | 35 |
| busi | 39 | x | | 5 | 29 |
| expect | 46 | x | | 4 | 23 |
| expand | 57 | x | | 3 | 17 |
| report | " | x | | 3 | 17 |
| spend | " | x | | 3 | 17 |
| feder | " | | x | 3 | 17 |
| unemploy | 90 | x | | 2 | 11 |
| interest | 112 | x | | 1 | 15 |
| administr | " | | x | 1 | 15 |
| next | " | x | | 1 | 15 |

Figure 2-16: Suggestions that add relevant documents for the seed query `Economic Projections`

precision,

A term yielding a decrease in precision may not necessarily be a bad suggestion. Figure 2-17 displays terms suggested by the $DM_{nfx}$ algorithm which yield a decrease in precision when added to the query from Figure 2-3. The terms `fiscal` and `budget` are semantically related to the query but still decrease precision. Moreover, notice that terms `china` and `japan` can focus the query among alternative search paths. They happen not to improve the precision of the seed query because they do not narrow the query in the direction of the particular information need as described in English in the topic's description and narrative fields. These last two suggestions focus the query to specific countries of interest. However, the information need as stated in the TREC query specifically requests documents including "a projection of the value of an economic

| Term | + Docs | + % Docs | Term | + Docs | + % Docs |
|---|---|---|---|---|---|
| nation | -1 | -5 | between | -12 | -70 |
| product | -2 | -11 | world | -13 | -76 |
| improv | -2 | -11 | provid | -13 | -76 |
| deficit | -2 | -11 | premier | -13 | -76 |
| should | -3 | -17 | foreign | -13 | -76 |
| industri | -3 | -17 | fiscal | -13 | -76 |
| howev | -3 | -17 | expert | -13 | -76 |
| third | -4 | -23 | develop | -13 | -76 |
| suppli | -4 | -23 | social | -14 | -82 |
| polici | -4 | -23 | problem | -14 | -82 |
| figur | -4 | -23 | power | -14 | -82 |
| estim | -4 | -23 | plan | -14 | -82 |
| budget | -4 | -23 | loan | -14 | -82 |
| futur | -5 | -29 | germani | -14 | -82 |
| bank | -5 | -29 | enterpris | -14 | -82 |
| term | -7 | -41 | debt | -14 | -82 |
| senior | -7 | -41 | relat | -15 | -88 |
| privat | -7 | -41 | reform | -15 | -88 |
| gener | -7 | -41 | meet | -15 | -88 |
| export | -7 | -41 | japan | -15 | -88 |
| construct | -7 | -41 | intern | -15 | -88 |
| will | -8 | -47 | communist | -15 | -88 |
| invest | -8 | -47 | assist | -15 | -88 |
| govern | -9 | -52 | western | -16 | -94 |
| dam | -9 | -52 | program | -16 | -94 |
| cost | -9 | -52 | polit | -16 | -94 |
| condition | -9 | -52 | offici | -16 | -94 |
| would | -10 | -58 | minist | -16 | -94 |
| presid | -10 | -58 | fund | -16 | -94 |
| countri | -10 | -58 | financ | -16 | -94 |
| capit | -10 | -58 | european | -16 | -94 |
| build | -10 | -58 | cooper | -16 | -94 |
| billion | -10 | -58 | chines | -16 | -94 |
| auster | -10 | -58 | china | -16 | -94 |
| work | -11 | -64 | agricultur | -16 | -94 |
| million | -11 | -64 | visit | -17 | -100 |
| told | -12 | -70 | east | -17 | -100 |
| shortag | -12 | -70 | aid | -17 | -100 |
| help | -12 | -70 | | | |

Figure 2-17: Term suggestions that decrease precision for the seed query `Economic Projections`

indicator". The documents containing `china` or `japan` may not satisfy this requirement. These alternative search paths are not explicitly represented in the query, and thus it is unreasonable to penalize the refinement algorithm for showing suggestions that focus the query towards such paths. In addition, in Section 2.4.3 we demonstrated that suggestions that are semantically related to the query may decrease precision. Since we are unable to assess the usefulness of terms that do not improve precision, our precision improvement experiments only consider precision-enhancing term suggestions.

### 2.4.5 Runtime Measurements

This section presents experimental results comparing the average total execution times of the $DM_{nfx}$ and RMAP query refinement algorithms. Each query was refined by each

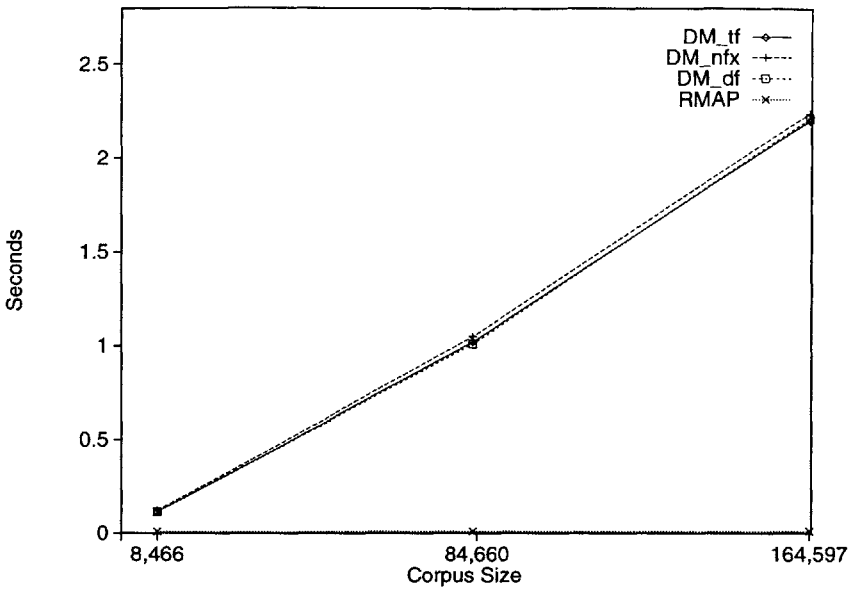



Figure 2-18: Timing performance for the DM and RMAP algorithms

algorithm a total of 5 times. The maximum and minimum runtimes were discarded, and the average of the remaining three measurements is reported below as the average execution time. All the execution time data were generated on a virtually unloaded Intel PC with a 75 MHz Pentium and 96MB of memory running the Linux operating system. The data are stored on a 32 GB Level 5 RAID system connected via an Ultra Fast and Wide SCSI bus.

Figure 2-18 illustrates the dramatic reduction in execution time that RMAP with $m = 100$ achieves versus the DM algorithms with $r = 100$. As suggested by the runtime analysis presented in section 2.3, the experimental data confirms that the DM algorithm is much more sensitive to corpus size than RMAP. For example, for a corpus of 84660 documents, RMAP suggests 100 terms in an average time of 12 *ms* over 150 queries, while the DM algorithm requires 1.05 *seconds* for the same task. This implies that on average, RMAP is capable of refining almost 88 queries in the amount of time it takes DM to refine one. For a corpus approximately twice as big, this ratio goes from 88:1 to

186:1.

### 2.4.6 Storage Requirements

The dramatic reduction in response time achieved by RMAP comes at the cost of increased storage space. This section presents the actual sizes of the DM and RMAP data structures for the implementations analyzed above. For DM, the size includes the size of the inverted file that maps terms to matching documents plus the size of the inverted file that maps documents to the terms they contain. For RMAP, the size includes the size of the inverted file mapping terms to the precomputed suggestions. All inverted files are kept hashed.

| | # of Docs in Collection | | |
|---|---|---|---|
| Algorithm | 8460 | 84660 | 164597 |
| $DM_{nfx}$ | 20 | 210 | 607 |
| RMAP | 41 | 135 | 804 |

Figure 2-19: Size (in MB) of Databases. DM versus RMAP algorithms

Table 2-19 shows the amount of storage required by $DM_{nfx}$ and RMAP with $m = 100$ for collections of different sizes. RMAP's fixed row size ($m$) allows a smaller rate of growth as the collection size increases. Notice that RMAP needs the data structures required by $DM_{nfx}$ plus an additional table mapping query terms to term suggestions. In this experiment RMAP accounts for a 30% increase in storage space. For a more detailed explanation of the data structures generated by the indexing process the reader is referred to Section 4.5.

## 2.5 Summary

Query refinement is an essential information retrieval tool that interactively recommends new terms related to a particular query. This chapter proposes an experimental frame-

work for evaluating whether an algorithm suggests effective query refinements and describes RMAP, a new algorithm that computes suggestions of similar quality to computationally intensive approaches. Our accuracy and runtime measurements of RMAP suggest that it is an attractive query refinement algorithm in situations where processing time is at a premium.

Further evaluations of RMAP and DM are necessary. For example, it is important to establish the effects of different choices for parameters such as RMAP's row size and the number of documents considered in DM's MATCH DOC phase. These parameters allow an administrator to fine tune a system based on performance and space constraints.

We are also investigating enhancements to RMAP as well as new query refinement algorithms. For example, we are exploring new ways to combine and rank the suggestions generated by RMAP including taking into account term proximities. An alternate refinement algorithm can take into account the number of documents that contain terms from the set of suggestions. Finally, we are interested in embedding query refinement algorithms in scalable hierarchical information systems using lossy index compression techniques.

# Chapter 3

# Coverage Query Refinement Using Query Lookahead

## 3.1 Introduction

Chapter 2 introduced fast and effective algorithms for query refinement as well as techniques for measuring the effectiveness of these algorithms. We also showed that at the heart of a query refinement algorithm is the method employed to select the terms to be displayed to a user as suggestions. Term selection is performed by all of the algorithms discussed so far by weight functions that take into account **individual** properties of terms. To the best of our knowledge, no previous algorithm considers **group** properties of terms. This chapter explores whether considering such group properties for term selection can increase the quality of the selected terms.

The central thesis of this chapter is that coverage term selection can enhance the effectiveness of term selection by minimizing the amount of redundancy and information loss in the selected set of terms. This goal should be attained without significant negative impact on concept recall and precision improvement. Our thesis is supported by experimental results using the experimental metrics developed in Chapter 2 as well as additional experimental metrics specifically designed to measure the effectiveness of our

new *coverage term selection* algorithms.

One important objective of a set of term suggestions is to offer a user a succinct set of choices for refining the seed query. Short queries can often be refined in multiple ways. For instance, consider the seed query `video`. It is difficult to determine whether the user is really interested in video compression formats, video servers, streaming video over the Internet or video retrieval systems. Lacking the information necessary to predict the more specific information need of interest, the system should offer a set of refinement choices capable of narrowing the scope of the seed query around any potential subarea of interest. A precise definition of succinctness must be developed before any algorithms for selecting succinct query suggestions can be analyzed. This chapter presents a model of ideal term selection that formalizes these intuitions. The model conjectures that an ideal set of terms should induce a set of corresponding refined queries whose result sets approximate a partition of the seed query's result set.

A term selection algorithm should avoid the *information loss* that may result from selecting a set of terms pertaining exclusively to a small fraction of the documents in the seed result set. Otherwise, significant portions of the result set may remain unnoticed or even inaccessible to users via the user interface mechanisms provided. The selected terms should provide refinement paths to as many of the documents in the seed result set as possible. A simple way to avoid information loss is to select terms that *appear* in, or *cover*, as many of the documents in the seed result set as possible. This is the origin of the name *coverage term selection* to describe algorithms that attempt to maximize coverage and as a result minimize information loss.

It is not difficult to select terms that achieve low information loss if one does not care about term redundancy. In Section 3.4 we will demonstrate experimentally that an algorithm can achieve low information loss simply by selecting the most common terms. Such terms, however, are unlikely to be useful for query refinement because they achieve very little search space reduction when combined with the seed query. The result set of a seed query combined with a very common term tends to be very similar to the

result set of the seed query by itself. Thus, such common terms are redundant in the sense that they offer very similar refinement paths. An ideal term selection algorithm should reduce information loss without achieving very high redundancy. A simple way to maintain redundancy low is to avoid selecting terms that generate very similar result sets. A key benefit of coverage term selection is its ability to cope with term redundancy and information loss by using information about the result set induced by each candidate term suggestion.

This chapter introduces the technique of *query lookahead* and its application to efficiently perform coverage term selection. *Query lookahead* is the process of eagerly evaluating multiple refined queries that are automatically generated from an initial *seed query*. The refined queries are generated by combining terms extracted from the documents that match the seed query. Term selection chooses from the set of query refinement terms those terms that provide to the user the most effective means for improving the seed query. The result sets of the refined queries serve as input to the term selection algorithm.

It will also be shown in this chapter that ideal term selection is an NP-complete problem. This is not surprising since the number of possible subsets of terms that could potentially be selected grows exponentially with the number of terms. The number of terms in turn is proportional to the size of the result set. This chapter will introduce *coverage term selection* (*CTS*), a greedy approximation coverage term selection algorithm, as well as experimental measurements comparing the effectiveness of *CTS* with that of $DM_{nfx}$, an algorithm introduced in Chapter 2 representing the class of algorithms performing term selection based exclusively on individual frequency-based properties of terms. *CTS* runs in $\mathcal{O}(\|\mathcal{F}(\mathcal{D}^*(q))\| \cdot w^2 \cdot \|\mathcal{D}^*(q)\|)$ where $w$ is the number of terms that must be selected, $\mathcal{D}^*(q)$ is the sample of the result set from which terms are extracted, and $\mathcal{F}(\mathcal{D}^*(q))$ is the set of candidate terms.

Briefly, the experimental evidence presented in this chapter supports our central thesis that the information about group properties of terms, and in particular the information

about the result sets of refined queries, can be exploited to select terms that reduce information loss while maintaining low redundancy. Our experiments also demonstrate that query lookahead can be used to implement coverage term selection efficiently. Experiments with both concept recall (Section 2.4.2) and precision improvement (Section 2.4.3) show that coverage term selection generates terms that are semantically related to the query and increase precision. The experiments also suggest that *CTS* is too slow to be used in practice at least for large result sets (500 documents), but could be used for smaller sets of documents. A hybrid algorithm that we call *COMBO* can reduce the negative impact on concept recall and precision improvement and reduce running time significantly while at the same time achieving significantly less redundancy and information loss than $DM_{nfx}$.

The remainder of this chapter is organized as follows:

Section 3.2 presents the formal model for coverage term selection.

Section 3.3 describes a practical greedy query refinement algorithm using coverage term selection.

Section 3.4 presents experimental results illustrating the impact of query lookahead on term selection effectiveness.

Section 3.5 summarizes the major findings of the chapter.

## 3.2 Term Selection Criteria

Throughout this dissertation we use the phrase *search terms* to denote the words, phrases or other properties by which documents can be looked up in an information retrieval system. One very common class of search terms is the words that appear in the text of documents. Other examples of search terms include document dates, authors and references. Search terms are in essence the simplest possible query to an information

retrieval system. A document that *contains* a search term matches the query consisting of that term.

Some definitions in this section will assume the existence of an underlying document corpus or universe which will be denoted $\mathcal{C}$.

In this section, a query is defined to be a Boolean combination of terms. Boolean queries may strike the reader as over restrictive, in particular, because this query model offers no natural mechanism to rank matching documents according to their belief of relevance. Strictly Boolean queries assign each matching document a Boolean value of true, each non-matching document a value of false, and make no further distinction between different matching documents. Our choice of query model, however, is strictly for convenience and simplicity. Our results apply to other query languages, including the vector query model. In Section 3.3 we present an implementation of coverage term selection that uses an extended Boolean query model supporting document ranking.

**Definition 3.1 (Query).**

*A* **query** *can be recursively defined as follows:*

- *A search term $f$ is a query* **matching** *all documents containing at least one occurrence of the search term.*

- *If $p$ and $q$ are queries, then $p \wedge q$ is a query matching all documents which match* **both** $p$ **and** $q$. *The $\wedge$ operator is called the* **conjunction** *operator.*

- *If $p$ and $q$ are queries, then $p \vee q$ is a query matching all documents which match* **either** $p$ *or* $q$ *or both. The $\vee$ operator is called the* **disjunction** *operator.*

- *If $p$ is a query, then $\neg p$ is a query matching all documents which* **do not** *match $p$. The $\neg$ operator is called the* **negation** *operator.*

The following definition formalizes the by now familiar concept of result set and introduces term sets.

**Definition 3.2 (Result Sets and Term Sets).**

- *The* **result set** *of a query $q$, denoted $\mathcal{D}(q)$, consists of all the documents from the underlying corpus $\mathcal{C}$ that match $q$.*
- *The* **term set** *of a set of documents $D$, denoted $\mathcal{F}(D)$, consists of all the search terms appearing in at least one document from $D$. $\mathcal{F}(q)$ is shorthand for $\mathcal{F}(\mathcal{D}(q))$.*

A large number of term selection methods have been proposed in the past. All of these previous methods only consider individual properties of terms. For instance, the term weighting functions used in Chapter 2 to select terms for query refinement assign a weight to each term based on the frequency of occurrence of the term in the result set documents as well as in the entire corpus. Due to their emphasis on term frequency information, such methods may end up selecting redundant terms or terms that induce information loss. Intuitively, two or more terms are redundant when they pertain to the same specific topic. Information loss occurs when one or more of the result set documents is not represented by any of the refinement terms.

Figure 3-1 further illustrates the concepts of information loss and redundancy. It shows a seed query $q$ and two refined queries $q_1$ and $q_2$ obtained by conjoining each of two selected terms to the seed query. Conjunction ensures that the result sets for the refined queries only comprise those documents from the seed result set that contain each suggested term (i.e. $\mathcal{D}(q_i) \subseteq \mathcal{D}(q)$). Shown are Venn diagrams (a), (b) and (c) representing three possible relationships between the three result sets in question. Diagram (a) shows how the union of the result sets of $q_1$ and $q_2$ include most of $\mathcal{D}(q)$, thus yielding high *coverage*. However, the result sets for the refined queries contain a significant number of documents in common, thus yielding high *overlap*, and thus high redundancy. Another suboptimal possibility is depicted in diagram (b). Here the result sets of the refined queries have no overlap, but achieve significantly less coverage, thus yielding higher information loss. Ideally, one would want to select terms inducing queries whose result sets resemble diagram (c) in which the result sets of the induced queries achieve **both**

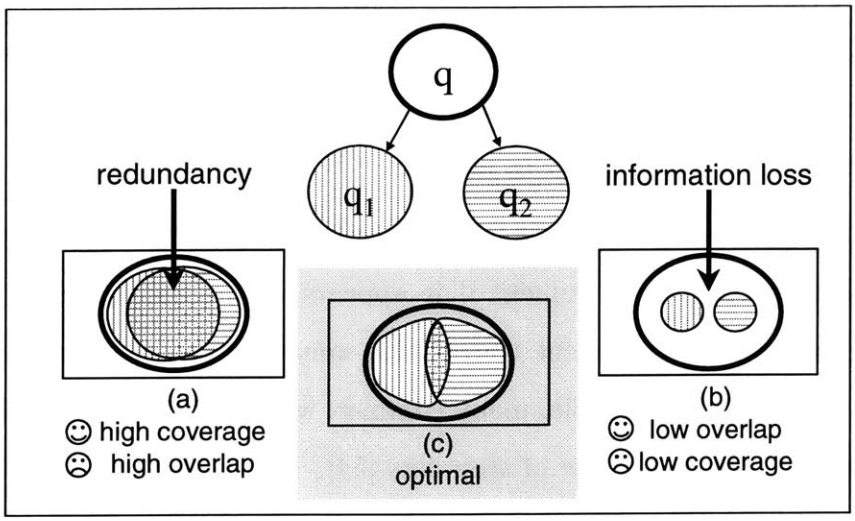



Figure 3-1: A query hierarchy and Venn diagrams depicting three possible relationships between the result sets of an initial query and two refined queries.

low redundancy (low overlap) **and** low information loss (high coverage).

One critical objective of an effective set of query suggestions is to present a succinct set of choices for refining the seed query. The query refinement paradigm introduced in Section 2.1 assumes that users submit short under-specified queries hoping that the query refinement system will guide them through the process of focusing their queries on more specific information needs. From the perspective of the system it is in general difficult to exactly predict a user's specific information need from a short seed query. The information necessary to make this prediction is simply missing from the seed query. For instance, consider the seed query `video`. It is difficult to determine whether the user is really interested in video compression formats, video servers, streaming video over the Internet or video retrieval systems. Each of the more specific subareas of interest represents a potential *search path* that a user may follow on his/her way to focusing the query. Thus, while an under-specified seed query may be refined along a number of possible search paths, the system is in general incapable of predicting the particular search path of interest to the user.

The next best thing to predicting the user's search path is to offer the user as many of the available search paths as possible. This can be achieved by representing possible search paths in term sets that minimize both redundancy information loss. We will now examine each of these two objectives in turn.

The discussion that follows will be based on the following simplifying assumption. A search term *describes* a document if it appears in the document. Clearly, not all terms that appear in a document convey the same amount of information about the document's content. For example, many common words are ignored by the information retrieval system through the use of stop-lists [54]. However, the bulk of terms that do not appear in stop-lists can be expected to describe the documents that contain them.

*Information loss* may result from selecting a subset of terms from the seed query's term set that does not describe a significant portion of the seed query's result set. As a result, one or more of the available search paths may not be offered to a user simply because the documents that embody the search path may not be represented by the selected terms. A set of terms that minimizes information loss may be computed by making sure that the selected terms appear in as many of the documents in the result set as possible.

*Redundancy*, on the other hand, may result from selecting terms representing the same search paths. Clearly, as the number of redundant terms increases, so does the number of redundant search paths that will be offered to the user. Redundant terms could be avoided by making sure that terms appearing in very similar subsets of result set documents are not simultaneously selected. One possibility is to minimize the intersection, or overlap, between the sets of documents where each pair of selected terms appear. A simple example of redundant terms is synonyms. Even this seemingly obvious case would be difficult to cope with without information about the specific documents in the result set where each term appears.

From Figure 3-1 it should be easy to see that when the terms partition the seed result set both redundancy and information loss are minimized. Our goal is to define a function

that measures how well a set of candidate terms partition the seed result set. We call such a function a *partition difference function.* The problem of finding an ideal set of terms will be formulated as a minimization problem of a partition difference function encompassing both minimal information loss and minimal redundancy. We will define our partition difference function using random variables [18].

For simplicity, we will first consider the optimization problem in its most general setting before we apply it to queries and result sets. Consider a set $S$, corresponding to the seed result set $\mathcal{D}(q)$, and a set $C$ of subsets of $S$, corresponding to the set of result sets for some subset of the child queries of $q$. Using query conjunction, $C$ could be defined as follows:

$$C = \{\mathcal{D}(q \wedge f) | f \in F\}$$

One possible partition difference function can be formulated by first defining a discrete random variable $x^{S,C}$ that measures, for a randomly selected element $s$ of $S$, the proportion of sets from $C$ where $s$ appears. That is,

$$x^{S,C} \equiv \frac{\|\{c|c \in C, s \text{ a random element from } S, s \in c\}\|}{\|C\|} \tag{3.1}$$

We use the notation $\|C\|$ to denote the cardinality of set $C$. Notice that the possible range of values for $x^{S,C}$ lies within the unit interval $[0, 1]$. The partition difference function can be simply defined as the average or expected value $E[x^{S,C}]$.

To a first approximation, the larger the value of $E[x^{S,C}]$ for a given $C$, the worse $C$ approximates a partition of $S$ due to a higher overlap among the sets in $C$. Thus, using this partition difference function, coverage term selection could be formulated as a minimization problem. However, there is one caveat. Consider the extreme case in which the randomly selected element from $S$ appears in none of the subsets in $C$. In this case, $x^{S,C} = 0$. Thus, such an element lowers $E[x^{S,C}]$, but increases the gap between $C$ and a partition of $S$. Thus some kind of "penalty" should be introduced when elements in $S$ do

not appear in any of the subsets in $C$. This penalty corresponds to our previous notion of information loss. We are now in a position to give a formal definition of partition difference that incorporates an information *loss-penalty.*

**Definition 3.3 (Partition Difference Function).**

$$\begin{aligned} \textit{Let} &\equiv \textit{a set} \\ C &\equiv \textit{a collection of subsets from } S \\ s &\equiv \textit{element randomly selected from } S \\ n &\equiv \|\{c|c \in C, s \in c\}\| \end{aligned}$$

*and let $x^{S,C}$ be a random variable (RV) defined as follows:*

$$x^{S,C} \equiv \begin{cases} \frac{n}{\|C\|} & : \quad n \neq 0 \\ loss - penalty & : \quad n = 0 \end{cases} \tag{3.2}$$

*Then the* **partition difference function** $\mathcal{P}(S, C)$ *is defined as* $E[x^{S,C}]$.

The value of *loss-penalty* in Equation 3.2 is implementation dependent and should reflect the relative tolerance to information loss. The larger the value chosen for *loss-penalty*, the smaller the tolerance to information loss. In any case, the value of *loss-penalty* should always be strictly greater than $\frac{1}{\|C\|}$ because otherwise uncovered documents will decrease partition difference, which is the value that must be minimized.

We can consider two extreme cases that intuitively validate our mathematical model of partition difference. In one extreme case the subsets in $C$ are all equal to $S$, therefore maximal overlap is achieved. The corresponding partition difference is equal to 1. When the subsets in $C$ perfectly partition $S$ both maximal coverage and minimal overlap are achieved. The corresponding partition difference is $\frac{1}{\|C\|}$, its minimum possible value when *loss-penalty* $> \frac{1}{\|C\|}$ . Note that a perfect partition difference of 0 is unattainable when the *loss-penalty* is greater than zero.

A example can further illustrate our definition of partition difference. Figure 3-2

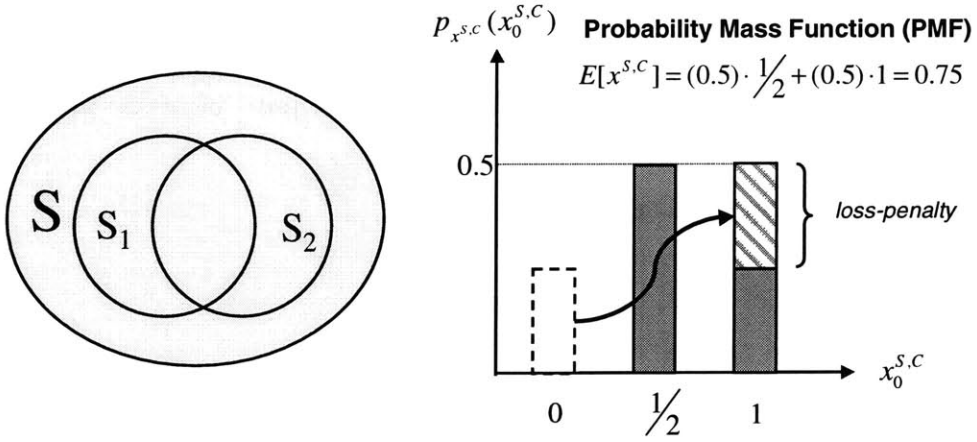



Figure 3-2: Venn diagram illustrating the partition difference ($E[x^{S,C}]$) of two sample subsets $S_1$ and $S_2$ of a set $S$ assuming *loss-penalty* $= 1$, $\|S_1\| = \|S_2\|$, $\|S_1 \bigcup S_2\| \div \|S\| = 0.75$, $\|S_1 \bigcap S_2\| \div \|S_1\| = 0.5$

shows the Venn diagram for an example set $S$ and two subsets $S_1$ and $S_2$. Set $C$ includes $S_1$ and $S_2$. The example has been carefully constructed to make it easy to visualize the probability mass function (PMF) for $x^{S,C}$ and to compute the expected value $E[x^{S,C}]$. The PMF is shown on the right side of the Figure. Notice that the *loss-penalty* of 1 effectively moves the chunk of the histogram that would have corresponded to $x^{S,C} = 0$ to the column corresponding to $x^{S,C} = 1$. As a result, the expected value drifts to the right to take into account the amount of information lost by $C$.

Although other approaches to define partition difference functions are quite conceivable, our approach using random variables has several desirable properties. First, as was illustrated with the example in Figure 3-2, it is relatively simple and intuitive to visualize the PMF of $x^{S,C}$ from the corresponding Venn diagram. Second, the PMF provides for a useful visual aid for comparing the partition difference achieved by alternative choices of $C$. Third, our approach permits the easy computation of the partition difference function by considering every element of $S$ in sequence and determining the number of sets in $C$ where it appears.

We can now apply our notion of partition difference to the original problem of coverage

term selection. A definition of optimal term selection simply says that for a set of terms to be optimal, the terms must optimize the partition difference of their corresponding result sets relative to the seed result set. Definition 3.4 puts all of the pieces together.

**Definition 3.4 (Optimal Term Selection).**

$$\begin{aligned} \textit{Let } q &\equiv \textit{seed query} \\ S &\equiv \mathcal{D}(q) \\ F &\equiv \textit{candidate subset of terms from } \mathcal{F}(q) \\ w &\equiv \textit{number of terms to be selected} \\ C &\equiv \{D(q \wedge f) | f \in F\} \\ C^* &\equiv \{D(q \wedge f) | f \in \mathcal{F}(q)\} \\ 2^{C^*} &\equiv \textit{Power set of } C^* \end{aligned}$$

*we say $F$ is optimal if and only if*

$$\mathcal{P}(C, \mathcal{D}(q)) = \min_{C' \in 2^{C^*} \wedge \|C'\| = w} \mathcal{P}(C', \mathcal{D}(q)) \tag{3.3}$$

Equation 3.3 deserves some explanation. The right hand side represents the minimum partition difference attainable by any subset of terms from the term set $\mathcal{F}(q)$. The set $C$ is the set of result sets induced by conjoining the candidate terms in $F$. $C^*$ is the analogous set but for the entire term set $\mathcal{F}(q)$. The sets $C$ and $C^*$ are necessary because we have decided to define partition difference in terms of sets of results sets instead of sets of terms. The left hand side represents the partition difference attained by the candidate terms. So, the candidate terms are optimal if and only if they attain the minimum partition difference attainable by some subset of terms derived from the result set of $q$.

In order to understand the computational complexity of the coverage term selection, it is again convenient to abstract out queries and their result sets and reformulate the problem of finding an optimal set of terms as that of finding an *optimal set partition.* In the style of [23], Definition 3.5 provides the decision version of this minimization problem.

**Definition 3.5 (OPTIMAL-SET-PARTITION Problem).**

**INSTANCE:** $< S, C, k, \alpha, l >$

*A collection $C$ of subsets of a finite set $S$, an integer $k$, an integer $l$, a real number $\alpha \in [0, 1]$.*

**QUESTION:** *Does $C$ contain a subcollection $C' \subseteq C$ with $\|C'\| \leq k$ such that $\mathcal{P}(C', S) \leq \alpha$ assigning a value of $l$ to the loss-penalty?*

Theorem 3.1 states that this problem is NP-complete. The known NP-complete problem EXACT-COVER-BY-3-SETS (X3C) [24] takes a set $S$ and a collection $C$ of subsets of 3 elements and determines whether their is a subset $C^*$ of $C$ such that every element of $S$ appears in exactly one subset in $C^*$. On the other hand OPTIMAL-SET-PARTITION attempts to select the set $C$ of subsets that best approximates a partition of a set $S$. So at first sight it seems that the OPTIMAL-SET-PARTITION is an generalization of, and thus a harder problem than, X3C.

**Theorem 3.1.** *OPTIMAL-SET-PARTITION is NP-complete*

*Proof.* As is customary for NP-completeness proofs, we will split the proof in two parts. The first part proves that the problem is in NP. The second part proves that a known NP-complete problem can be reduced to OPTIMAL-SET-PARTITION in polynomial time (NP-hardness).

*Part 1:* Show OPTIMAL-SET-PARTITION $\in$ NP

For this part of the proof, it suffices to show that for any given set $C$ of subsets of $S$, it is possible to compute the partition difference $\mathcal{P}(C, S)$ in polynomial time. If this is the case then the result follows since a nondeterministic algorithm could just guess a solution $C' \subseteq C$ and determine whether the solution had partition difference greater or equal to $\alpha$.

The partition difference of such a guessed solution can be computed in two steps. The first step computes the PMF for the random variable $x^{S,C}$ and the second

step computes its expected value. The PMF can be computed by examining each element $e$ of $S$ in turn and determining the value of the random variable if $e$ had been the randomly chosen element. The PMF will consist of the histogram containing the relative frequency of each possible value of the random variable.

As shown in Figure 3-2, computing the expected value $E[x^{S,C}]$ can be easily done in polynomial time using the PMF by accumulating the products of the relative frequencies times the corresponding values of the random variable.

*Part 2:* Show OPTIMAL-SET-PARTITION $\in$ NP-hard

We will show that X3C can be reduced in polynomial time to OPTIMAL-SET-PARTITION. Given an instance of X3C $< S, C >$ we construct an instance $< S', C', k', \alpha, l >$ of OPTIMAL-SET-PARTITION as follows:

$$\begin{aligned} S' &= S \\ C' &= C \\ k' &= \|C\| \\ \alpha &= \frac{1}{\|C\|} \\ l &= \|C\| \end{aligned}$$

Intuitively, the idea is to make every set $S_i$ in $C$ have an analogous set $S'_i$ in $C'$ with exactly the same elements. By setting $k' = \|C\|$ we let the partition of $S'$ have any size. The combination of a value of $\frac{1}{\|C\|}$ for *alpha* and a value of $\|C\|$ for the *loss-penalty* guarantees that the subset $C'$ of $C$ partitions $S'$. Clearly this rather direct reduction can be accomplished in polynomial time since the only value not readily available is $\|C\|$.

The close resemblance between X3C and OPTIMAL-SET-PARTITION makes our proof rather straight forward. We must show that our reduction maps every X3C

instance into an OPTIMAL-SET-PARTITION instance. A key element in the reduction is setting $\alpha = \frac{1}{n}$ and $l = n$. These conditions can only be achieved by a collection of subsets forming a perfect partition of $S'$. By definition any subset of $C$ that satisfies X3C must form a partition of $S$. But, this can only happen if and only if the corresponding subset of $C'$ forms a partition of $S'$. This is true since every subset of $S$ in $C$ has a corresponding subset of $S'$ in $C'$ with exactly the same elements. Moreover, our construction does not add any elements to $S'$ not previously present in $S$.

□

Theorem 3.1 implies that if the partition difference of a set of terms is a good model of its succinctness, then no efficient (polynomial time) algorithm for ideal term selection exists unless $P = NP$. In Section 3.3.5 we will present $CFS$, a simple greedy approximation algorithm as well as experimental results measuring its effectiveness in minimizing both redundancy and information loss.

Reducing partition difference is not the only desirable property of an effective set of terms. Clearly, term weighting functions should also play a role in the term selection process. Other optimizations are also conceivable. Different desirable properties of term sets could be combined using weighted averages into a single objective function suitable for an optimization problem. However, it is not too difficult to see that as long as partition difference remains a factor in such an objective function the problem of optimal term selection remains NP-complete. Since we believe that minimal partition difference is an essential ingredient of term selection, the remainder of this dissertation will focus on efficiently approximating optimal term selection based primarily on this objective function.

## 3.3 Term Selection Algorithms

We present in this section a practical algorithm for query refinement that exploits the information provided by a query lookahead algorithm to implement coverage term selection. To accomplish this objective, many details that were omitted in the model presented in the last section must be specified. For instance, a practical algorithm must establish reasonable bounds on the amount of time and computation used. This is especially true for query refinement, since, due to its expected use, response time must be kept within a limit tolerable by an interactive user. The section begins with an overview of the query refinement algorithm as a sequence of phases. Each phase is subsequently described in detail in a separate subsection. The descriptions of the algorithms will be based on the definitions introduced in Section 3.2.

### 3.3.1 Overview of the Query Refinement Algorithm

As shown in Figure 3-3, our query refinement algorithm can be conceptually decomposed into four phases: *result set reduction*, *term extraction*, *query lookahead* and *term selection*. The input to the algorithm is the seed user query $q$ with its corresponding result set $\mathcal{D}(q)$ and an integer $w$ specifying the number of query suggestions that the algorithm should generate. The output of the query refinement algorithm consists of a set $F^*$ containing the $w$ selected term suggestions.

The first phase of query refinement, called *result set reduction*, computes a sample $\mathcal{D}^*(q)$ of documents from $\mathcal{D}(q)$ that will be scanned for terms. The second phase of query refinement, called *term extraction*, computes the term set $\mathcal{F}(\mathcal{D}^*(q))$ and constructs, for each extracted term $f_i$, a query $q_i = (q \wedge f_i)$ by conjoining the term with the seed query $q$. *Query lookahead*, the third phase, outputs the result set of each $q_i$. The result sets of these conjunctive queries constitute input to the fourth and last query refinement phase, *term selection*. *Term selection* computes the set of terms that will be offered to a user as query refinement suggestions. $\mathcal{D}(q)$.

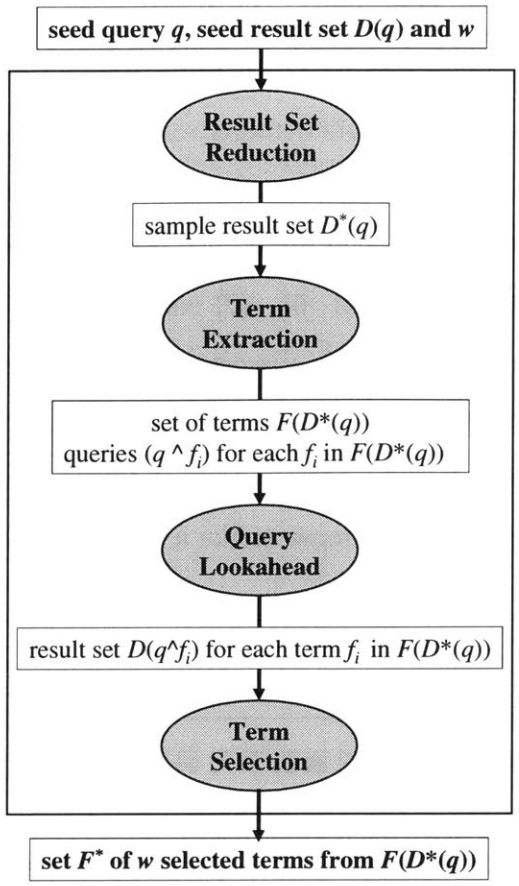



Figure 3-3: Four conceptual phases of query refinement

For simplicity, Figure 3-3 only shows one term selection phase after the query lookahead phase. Additional term selection phases are quite conceivable. For instance, extremely common terms could be dropped right after the extraction phase before knowing their corresponding result sets. In Section 3.4 we experimentally demonstrate that such filtering can substantially reduce the runtime of coverage term selection. Our experiments also demonstrate that there should be at least one term selection phase that can use the result set information provided by the query lookahead phase to reduce information loss.

We will now discuss each query refinement phase in more detail.

### 3.3.2 Result Set Reduction

*Result set reduction* chooses a sample $\mathcal{D}^*(q)$ of documents from the seed result set $\mathcal{D}(q)$ that will be scanned for terms. We will demonstrate both theoretically (Section 3.3.2) and experimentally (Section 3.4) that the size of the result set sample plays a critical role in the performance of a query refinement algorithm. Our experience implementing previous systems supporting query refinement [68, 67] confirms the more recent theoretical and experimental results presented in this dissertation. While scanning a large sample of documents may result in an intolerably slow algorithm, scanning a small number of documents may result in a selection of terms that fail to describe an important part of the result set. Result set reduction must strike a balance between these two undesirable extremes.

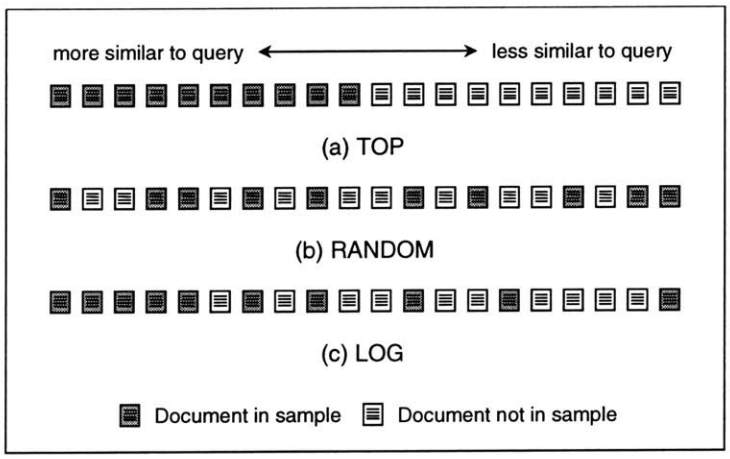



Figure 3-4: Three result set reduction methods

Three possible algorithms for result set reduction are depicted in Figure 3-4: TOP, RANDOM and LOG. Each algorithm in the figure selects a sample of ten documents from a result set of twenty. The documents are assumed ranked left to right by decreasing similarity to the seed query. However, only two of the algorithms require document ranking, namely TOP and LOG.

TOP is the algorithm that we have used in all our previous query refinement systems

[58, 68, 67] as well as in all the experiments presented in this dissertation. Due to time constraints we leave the implementation and measurement of RANDOM and LOG for future work.

TOP selects the contiguous subset of highest ranked documents. This algorithm has the advantage that it assigns higher weights to documents that are ranked higher by the query. It has the drawback that it completely ignores documents ranked lower. This may cause important large subsets of the result set to go unnoticed in the term set.

RANDOM weights all documents in the result set equally. As its name suggests, RANDOM picks each document with equal probability. Every segment of the ranked result set is represented in the sample with high probability. However, the set of documents ranked higher, which is likely to include documents more related to the query, does not receive any preference.

LOG attempts to get the best of both TOP and RANDOM algorithms. LOG selects a sample in which the number of unselected documents between selected documents increases logarithmically with the distance from the top ranked document. By placing more weight on documents ranked higher, LOG increase the chances that most documents that are definitely related to the query will be represented in the hierarchy. Documents ranked lower, on the other hand, are not ignored and do receive some weight.

### Time complexity of result set reduction

We will analyze the time complexity of this and the other three query refinement phases using the size of the seed result set and the size of the result set sample as the appropriate measures of input size. Further analysis would require information on the frequency distribution of queries which is not available to us. It is important to keep in mind that the size of the seed result set does depend on the size of the corpus. More specifically, if we assume that the frequency distribution of terms is independent of corpus size, then we should expect a linear increase in result set size as the corpus size increases.

Due to time and space limitations, the experimental results that will be presented in

this dissertation will all use the TOP algorithm. We leave the analysis of and experimentation with the other result set reduction approaches for future work. We now proceed to analyze the time complexity of TOP.

Computing a result set sample using TOP entails two steps: computing a document weight function for each document in $\mathcal{D}(q)$, and selecting the highest weighted documents. Most common document weighting functions can be computed in $\mathcal{O}(1)$ time. Therefore the time complexity of result set reduction is:

$$T_{Reduction} = \mathcal{O}(\|\mathcal{D}(q)\|) + \mathcal{O}(\|\mathcal{D}^*(q)\| \tag{3.4}$$

$$= \mathcal{O}(\|\mathcal{D}(q)\| \tag{3.5}$$

### 3.3.3 Term Extraction

The input to the term extraction phase includes the seed query $q$ and the sample $\mathcal{D}^*(q)$ of documents computed by result set reduction. Term extraction accomplishes two tasks. First, it computes the set $\mathcal{F}(\mathcal{D}^*(q))$ of terms appearing at least once in at least one of the sample documents. Our implementation avoids having to parse each document from scratch by keeping a table, called the document file, mapping document identifiers to the pre-parsed set of terms that they contain. Second, term extraction computes a set of queries $q_i$ of the form $(q \wedge f_i)$ by conjoining each term $f_i$ in $\mathcal{F}(\mathcal{D}^*(q))$ with the seed query $q$. Once evaluated by the query lookahead phase the result sets of these queries will be used by the term selection phase to select terms that minimize redundancy and information loss.

Term extraction is also responsible for keeping track of statistical term occurrence information. For instance, term extraction remembers the number of documents in the result set sample in which each term appears. Section 2.3 discussed several ways in which such information can be used to make inferences about the relative importance of terms for query refinement. Term occurrence information will be exploited by the term

selection phase.

### Time complexity of term extraction

The complexity analysis of term extraction is complicated by the fact that in general larger documents are more likely to match a given query. For simplicity, our analysis of term extraction will use the sum $\mu$ of the sizes of all the documents in the result set sample, as the appropriate measure of input size. More formally, $\mu$ can be defined as follows:

$$\mu = \sum_{d \in \mathcal{D}^*(q)} \|\mathcal{F}(\{d\})\| \tag{3.6}$$

Since the documents in $\mathcal{D}^*(q)$ must be scanned only once, the runtime complexity of term extraction is given by:

$$T_{extract} = \mathcal{O}(\mu) \tag{3.7}$$

Although this section presented term extraction as a phase separate from query lookahead, an implementation may not chose to separate the phases. The following section describes one way to combine both phases to obtain a more efficient query refinement algorithm.

## 3.3.4 Query Lookahead

Query lookahead, the third phase of query refinement, computes the result set of each query generated by term extraction. A naive query lookahead algorithm can evaluate each query $(q \wedge f_i)$ individually from scratch. However, the conjunctive nature of these queries yields result sets that are contained within the result set of the seed query. A naive algorithm can be improved by retrieving the set of documents containing the term $f_i$ from the inverted file and intersecting the corresponding set of documents with $\mathcal{D}^*(q)$. This

optimization reuses the previously computed result set of the seed query in computing the result set of the refined query.

**INPUT:**
Term set $\mathcal{F}(\mathcal{D}^*(q))$,
Set of queries $\{(q \wedge f_i) | f_i \in \mathcal{F}(\mathcal{D}^*(q))$

**PROCEDURE:**

(1) $F_0(q) = \emptyset$
(2) for each $f_i \in \mathcal{F}(\mathcal{D}^*(q))$ do
  (3) $D_0(q \wedge f_i) = \emptyset$
(4) $n = 0$
(5) for each $d \in \mathcal{D}^*(q)$ do
  (6) for each $f_i \in d$ do
    (7) $F_{n+1}(q) = F_n(q) \cup \{f_i\}$
    (8) $D_{n+1}(q \wedge f_i) = D_n(q \wedge f_i) \cup \{d\}$
    (9) $n = n + 1$

**OUTPUT:** Set of result sets $\{\mathcal{D}(q \wedge f_i) | \quad f_i \in \mathcal{F}(q)\}$

Figure 3-5: *LOOK*: A fast algorithm integrating term extraction with query lookahead

Further optimizations are also possible. Algorithm *LOOK* in Figure 3-5 achieves efficiency by combining term extraction with query lookahead. The algorithm is just a term extraction algorithm with additional bookkeeping code (steps (2-3) and (8) )to keep track of the set of documents where each extracted term appears. When step (8) encounters term $f_i$ inside document $d$, it adds $d$ to $\mathcal{D}(q \wedge f_i)$. In essence, *LOOK* builds an inverted index for the documents in the result set sample. Figure 3-6 presents a more graphical view of the operation of the algorithm. Notice that whenever a previously unseen term is encountered (**new** in the figure), *LOOK* creates a new category for the

corresponding query.

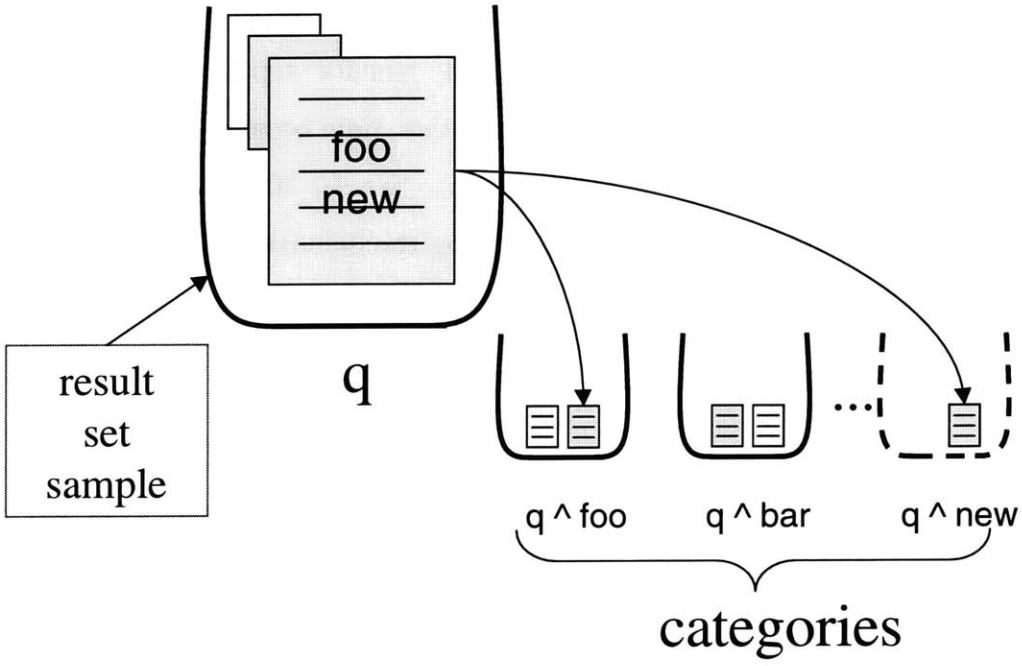



Figure 3-6: Diagram depicting the workings of the *LOOK* algorithm

*LOOK* is significantly faster than the optimized naive algorithm described above because computation is saved by not having to retrieve and parse into memory the inverted file entry for each extracted search term. This represents one less file system access per term. Our experience demonstrates that, even for small sample sizes, the number of extracted terms can be large. For the test collection used in the experiments of Section 3.4, a sample of one hundred documents generates an average of roughly seven thousand terms. Although caching can help, our experience suggests that parsing a large inverted file entry into an in-memory data structure is a relatively expensive operation and *LOOK* only needs to do it once for the seed query. Although we do not demonstrate it experimentally, we expect that *LOOK* will be significantly faster even if the entire inverted file fit in memory.

**Time complexity of query lookahead**

The computational complexity of *LOOK* is the same as that of term extraction since the loop in steps (2) and (3) can be accomplished in $\mathcal{O}(\|\mathcal{F}(\mathcal{D}^*(q))\|)$ and step (8) can be accomplished in constant time. This is obviously true for step (3), but not as obvious for step (8). Step (8) can be accomplished in constant time because the step always adds a previously non-existent document to the set $D_{n+1}(q \wedge f_i)$. Therefore, there is no need to check for duplicate elements. In summary, using the definition of $\mu$ in Equation 3.6, the complexity of *LOOK* is given by:

$$T_{LOOK} = \mathcal{O}(\mu) \tag{3.8}$$

*LOOK* is surprising in that it demonstrates that query lookahead does not require a qualitative impact on the runtime complexity of query refinement. There is no reason, at least theoretically speaking, not to take advantage of query lookahead to improve query refinement effectiveness. Section 3.4 will further demonstrate that this result is consistent with our experimental data.

### 3.3.5 Term Selection Algorithms

Query refinement algorithms must perform term selection in order to limit the number of terms suggested to the user. Intuitively, the refined queries should describe the parent result set as succinctly as possible. As explained in Section 3.2, term selection should pick terms representative of as many documents in the result set as possible. At the same time, term selection should minimize the number of redundant terms describing documents already represented by other selected terms. Term selection analyzes all the information obtained by the previous phases and attempts to select an optimal subset of term suggestions from $\mathcal{F}(\mathcal{D}^*(q))$ yielding minimal partition difference $\mathcal{P}$ (see Definition 3.3).

We know from Theorem 3.1 that computing an optimal subset of terms is a NP-

complete problem. Our goal for this dissertation is to find an efficient approximation algorithm that can select terms effectively. Our goal is not, however, to find the most effective nor the most efficient approximation algorithm. Rather, we focus on demonstrating that coverage term selection has potential to improve term selection effectiveness over previous algorithms that base their term selection decisions on individual statistical properties of terms. We leave the search for more efficient/effective algorithms for future work. Examples of algorithms using individual statistical properties of terms were studied in Chapter 2.

**INPUT:**

Set of result sets $\{\mathcal{D}(q \wedge f) | f \in \mathcal{F}(q)\}$

Integer $w$ specifying the number of terms to select

**PROCEDURE:**

(1) $idealSize = \|\mathcal{D}^*(q)\| \div w$

(2) Select $f \in \mathcal{F}(\mathcal{D}^*(q))$ such that $\|\mathcal{D}(q \wedge f)\| \approx idealSize$

(3) $n = 0$

(4) $C_0 = \mathcal{D}(q \wedge f)$

(5) $F_0^*(q) = \{f\}$

(6) while $n \leq w$ do

(7) Select $f \in \mathcal{F}(q)$ with maximal $\frac{\|\mathcal{D}(q \wedge f) - C_n\|}{\|\mathcal{D}(q \wedge f)\| + \|C_n\|}$

(8) $C_{n+1} = C_n \cup \mathcal{D}(q \wedge f)$

(9) $F_{n+1}^*(q) = F_n^*(q) \cup \{f\}$

(10) $n = n + 1$

**OUTPUT:** $< q, \mathcal{D}^*(q), \mathcal{F}^*(q), \{< q \wedge f, \mathcal{D}(q \wedge f) > | f \in \mathcal{F}^*(q)\} >$

Figure 3-7: *CTS*: A Greedy Approximation Algorithm for Coverage Term Selection

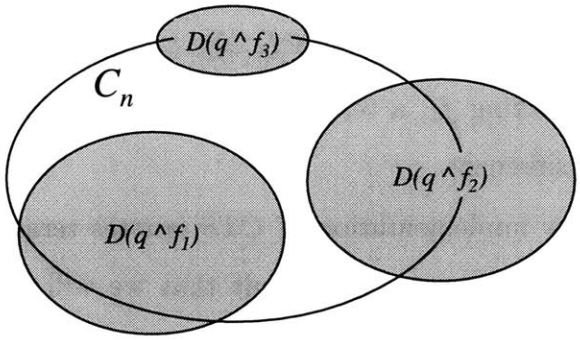
Algorithm *CTS* in Figure 3-7 is a starting point in the search for practical coverage term selection algorithms. It is a classic greedy algorithm [12] and it works by iteratively

selecting the terms that yield the highest "local" improvement in coverage with minimal overlap. Initially, the algorithm selects a term that is designed to break the seed result set into equal size pieces (steps (1) and (2)). If more than one term satisfies this condition, the algorithm selects the term with highest weight according to some term weighting function. The reader is referred to Section 2.3 for examples of term weighting functions. Frakes [21] and Salton [51] give comprehensive surveys of term weighting functions based on statistical term occurrence information that could be used by *CTS*.

The invariant of the loop in step (6) is that the set $C$ consists of the set of documents containing at least one of the terms in $F_n^*$ selected so far. We will refer to set $C$ as the *covered* documents and to the complementary set $\mathcal{D}^*(q) - C$ as the *uncovered* documents. Each iteration selects the term $f_i$ appearing in the largest proportion of uncovered documents (step 7). The algorithm tries to keep overlap to a minimum by normalizing the size of the set of uncovered documents added by $f_i$ by the sum of the sizes of the set of covered documents and the set of documents in $\mathcal{D}^*(q)$ where $f_i$ appears. Steps (8) through (10) restore the loop invariant. This iterative process continues until $w$ terms are selected.

The Venn diagram in Figure 3-8 illustrates how *CTS* selects terms that yield maximal "local" improvement in coverage while maintaining minimal "local" overlap. The figure represents a snapshot of *CTS* just before executing step (7) and after some number of iterations. At this time the algorithm must select a term from among three candidates $f_1$, $f_2$ and $f_3$ to extend $C_n$ to make $C_{n+1}$.

The figure includes the set $C_n$ of covered documents as well as the three sets $\mathcal{D}(q \wedge f_i)$ containing the documents where each candidate term $f_i$ appears. For illustration purposes we will assume that these four sets are configured as shown in the table underneath the Venn diagram in Figure 3-8. Each row in the table holds information about one set. The columns represent the name of the set (labeled $S$), the size of the set (labeled $\|S\|$), the overlap of the set with $C_n$ (labeled $\frac{\|S \cap C_n\|}{\|S\|}$) and the measure that the selected term must maximize (labeled $\frac{\|S - C_n\|}{\|S\| + \|C_n\|}$).




| $S$ | $\|S\|$ | $\frac{\|S \cap C_n\|}{\|S\|}$ | $\frac{\|S - C_n\|}{\|S\| + \|C_n\|}$ |
|---|---|---|---|
| $D(q \wedge f_1)$ | 50 | 0.9 | $\frac{5}{150} = 0.03$ |
| $D(q \wedge f_2)$ | 50 | 0.5 | $\frac{25}{150} = 0.17$ |
| $D(q \wedge f_3)$ | 10 | 0.5 | $\frac{5}{110} = 0.05$ |
| $C_n$ | 100 | 1.0 | $\frac{0}{200} = 0.00$ |

Figure 3-8: *CTS* selects the term maximizing $\frac{||\mathcal{D}(q \wedge f) - C||}{||\mathcal{D}(q \wedge f)|| + ||C||}$.

It can be seen from Figure 3-8 that *CTS* prefers $f_2$ over $f_1$ because it contributes more documents (25) to the covered set than $f_1$ (5). Also, *CTS* prefers $f_3$ over $f_1$ because, even though both contribute the same number of documents (5) to the covered set, $f_3$ yields less overlap (5 documents) than $f_1$ (45 documents). Although both $f_2$ and $f_3$ have the same fraction of new to covered documents (50%), *CTS* prefers $f_2$ because it adds a larger number of uncovered documents.

Simple mathematical analysis of the expression $\frac{||\mathcal{D}(q \wedge f) - C||}{||\mathcal{D}(q \wedge f)|| + ||C||}$ shows that this objective function will give preference to very common terms covering a large proportion of the result set. For instance imagine a term $f_4$ in Figure 3-8 such that $\mathcal{D}(q \wedge f_4)$ includes the union of all four sets depicted in the Venn diagram plus 10 more previously uncovered

documents. The size of $\mathcal{D}(q \wedge f_4)$ would be 145 and it would be contributing 45 previously uncovered documents. The value of $\frac{\|\mathcal{D}(q \wedge f) - C\|}{\|\mathcal{D}(q \wedge f)\| + \|C\|}$ would thus be $\frac{45}{245} = 0.18$. Therefore, *CTS* would end up selecting $f_4$, a term that, due to its high frequency, is not a good candidate for query refinement.

For this reason, our implementation of *CTS* ignores terms that cover more than a certain fraction of the result set. The result that we will present in Section 3.4, for instance, were generated by an algorithm limiting the proportion of result set documents covered by each term to $\frac{1}{2}$.

It was suggested to us recently [39] that an implementation of *CTS* that runs in linear time is possible. The idea is to maintain the sets $\mathcal{D}(q \wedge f) - C_n$ by removing items from it as we add the items to $C_n$. We are currently re-implementing *CTS* to incorporate this insight.

### Time complexity of *CTS* term selection

In this section we demonstrate that the expected runtime complexity of *CTS* is polynomial in the size of the result set sample $\|\mathcal{D}^*(q)\|$. Theorem 3.1 demonstrated that an ideal coverage term selection algorithm must run in time super-polynomial in the size of $\mathcal{F}(\mathcal{D}^*(q))$ unless $P = NP$. To simplify the analysis we will assume that a set union or difference operation between two of the input result sets in $\{\mathcal{D}(q \wedge f) > |f \in \mathcal{F}(\mathcal{D}^*(q))\}$ takes time $\mathcal{O}(\delta)$. We shall say more about $\delta$ below.

The time complexity of each step of *CTS* can be described as follows. Step (1) takes $\mathcal{O}(1)$ time and step (2) requires $\mathrm{O}(\|\mathcal{F}(\mathcal{D}^*(q))\|)$ time. Steps (3), (4) and (5) can all be done in constant $\mathcal{O}(1)$ time. The loop in step (6) is repeated $w$ times. During iteration $i$ of the loop, step (7) performs a set difference operation for each of $\|\mathcal{F}(\mathcal{D}^*(q))\| - (i-1)$ terms not yet selected. In the worst case, each set difference $\mathcal{D}(q \wedge f) - C$ is equivalent in complexity to $i-1$ set differences each taking $\mathcal{O}(\delta)$ time. Therefore, the complexity of step (7) is $\mathcal{O}((\|\mathcal{F}(\mathcal{D}^*(q))\| - (i-1)) \cdot (i-1) \cdot \delta)$. By a similar analysis, step (8) requires $\mathcal{O}(i-1)$ time. Finally, steps (9) and (10) only require $\mathcal{O}(1)$ time.

After comparing the complexity of all the steps, we see that the overall worst case time complexity of *CTS* is dominated by the loop in step (6). Therefore, the complexity of *CTS* is:

$$\begin{aligned}
T_{CTS} &= \sum_{1\leq i\leq w} [(\|\mathcal{F}(\mathcal{D}^*(q))\| - (i-1)) \cdot (i-1) \cdot \delta] \\
&= \sum_{1\leq i\leq w} [(\|\mathcal{F}(\mathcal{D}^*(q))\| - i + 1) \cdot (i-1) \cdot \delta] \\
&= \delta \cdot \sum_{1\leq i\leq w} \left[i \cdot \|\mathcal{F}(\mathcal{D}^*(q))\| - \|\mathcal{F}(\mathcal{D}^*(q))\| - i^2 + 2i - 1\right] \\
&= \delta \cdot \sum_{1\leq i\leq w} [\|\mathcal{F}(\mathcal{D}^*(q))\| \cdot (i-1)] - \delta \cdot \sum_{1\leq i\leq w} (i-1)^2 \\
&= \left[\delta \cdot \|\mathcal{F}(\mathcal{D}^*(q))\| \cdot \sum_{1\leq i\leq w} (i-1)\right] - \left[\delta \cdot \frac{(w-1)(w)(2w-1)}{6}\right] \\
T_{CTS} &= \mathcal{O}(\|\mathcal{F}(\mathcal{D}(q))\| \cdot w^2 \cdot \delta - w^3 \cdot \delta)
\end{aligned} \tag{3.9}$$

Equation 3.9 can be further simplified if we assume, as is usually the case, that the number of terms $w$ to be selected is smaller than the term set $\mathcal{F}(\mathcal{D}^*(q))$.

$$T_{CTS} = \mathcal{O}(\|\mathcal{F}(\mathcal{D}^*(q))\| \cdot w^2 \cdot \delta) \tag{3.10}$$

In practice, $\delta$ is not constant, but dependent on the size of the result set sample. More precisely, $\delta$ represents the sum of the sizes of the two sets taking part in the set operation. Lacking a precise probability distribution for the queries $(q \wedge f_i)$ it is not possible to obtain an average case bound. However, a worst case bound is possible. In the worst case, the size of the result sets $\mathcal{D}(q \wedge f_i)$ can be assumed to be equal to the document sample size. In this case we have $\delta = \mathcal{O}(\|\mathcal{D}^*(q)\|)$ and the complexity of *CTS* becomes,

$$T_{CTS} = \mathcal{O}(\|\mathcal{F}(\mathcal{D}^*(q))\| \cdot w^2 \cdot \|\mathcal{D}^*(q)\|) \tag{3.11}$$

The above complexity analysis confirms that the size of the result set sample and the size of its term set play a critical role in the time complexity of feature selection and that *CTS* is polynomial.

### 3.3.6 Time Complexity of Coverage Query Refinement

Adding the complexities of all the query refinement phases we get the complexity $T_{QR}$ of the complete query refinement algorithm.

$$\begin{aligned} T_{QR} &= T_{Reduction} + T_{LOOK} + T_{CTS} \\ &= \mathcal{O}(\|\mathcal{D}(q)\| + \mathcal{O}(\mu) + \mathcal{O}(\|\mathcal{F}(\mathcal{D}^*(q))\| \cdot w^2 \cdot \|\mathcal{D}^*(q)\|) \qquad (3.12) \\ & \qquad (3.13) \end{aligned}$$

Notice that the time for term extraction and query lookahead is included in $T_{LOOK}$. This analysis demonstrates that the time complexity of *CTS* is polynomial in the size of the seed result set, the size of the result set sample and the number of term suggestions that the algorithm must generate.

## 3.4 Evaluation of Algorithms

This section presents examples of terms suggested by the various term selection algorithms discussed so far followed by the results of four experiments comparing coverage term selection (*CTS*) with frequency-based term selection ($DM_{nfx}$). Our main objective is to quantitatively assess whether coverage term selection can improve the quality of query refinement terms and, if so, at what computational cost. The first experiment measures redundancy and information loss as defined in Section 3.2. The second and third experiments use the experimental metrics introduced in Chapter 2: concept recall (Section 2.4.2) and precision improvement (Section 2.4.3). The fourth experiment ex-

amines runtime efficiency. All four experiments use the same test collection of 84,660 Associated Press articles from 1989, and 150 test queries used in the experiments in Chapter 2 and described in detail in Section 2.4.1.

### 3.4.1 Experimental Setting

The four experiments reported in this section will analyze three experimental term selection algorithms: *CTS*, $DM_{nfx}$ and *COMBO*. *CTS* is the new coverage term selection algorithm described in Section 3.3. We selected $DM_{nfx}$ because it achieved the best concept recall and precision improvement among all other frequency-based algorithms considered in Chapter 2. *COMBO* combines coverage term selection (*CTS*) with frequency-based ($DM_{nfx}$) term selection. *COMBO* first uses $DM_{nfx}$ to reduce the set of candidate terms to twice the number of terms that must be selected in the end. *COMBO* then relies on *CTS* to select the final terms from the reduced set of candidate terms. Like $DM_{nfx}$, *COMBO* is fast because is has fewer candidate terms from which to choose. However, it may not be as effective as *CTS* in dealing with information loss and redundancy since it discards many terms before examining their result sets. *COMBO*, however, can behave as either $DM_{nfx}$ or *CTS* by controlling the number of candidate terms provided by *COMBO* to the *CTS* phase.

Each experiment will be conducted under four experimental configurations labeled A, B, C and D in Figure 3-9. Each configuration represents a different combination of *document frequency limit* with *result set sample size*. The document frequency limit determines the maximum document frequency allowed for the selected terms. The document frequency of a term $t$ is the proportion of documents in the corpus $\mathcal{C}$ containing $t$ (i.e. $\frac{\|\mathcal{D}(term)\|}{\|\mathcal{C}\|}$). As will be demonstrated in Section 3.4.6, reducing the number or terms that must be evaluated by a term selection algorithm can significantly reduce the running time of the algorithm. One possible way to reduce the number of candidate terms is to discard terms with very high document frequencies. Such terms appear in a large fraction of the documents in the corpus and are less descriptive and thus less useful for

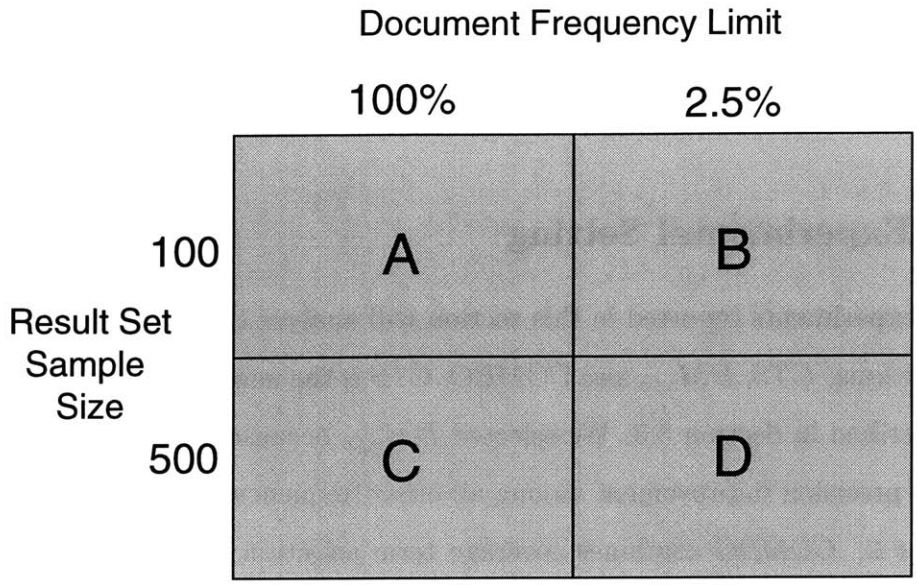



Figure 3-9: Experimental configurations under which term selection algorithms are compared

query refinement purposes. Our experiments assess the impact that such term filtering may have on the quality of the selected terms. Configurations A and C in the left column of Figure 3-9 use a document frequency limit of 100% which means that no candidate terms are discarded before the term selection phase. Configurations B and D in the right column of Figure 3-9 use a document frequency limit of 2.5%. Although 2.5% may sound overly restrictive, in our test collection of eighty-four thousand documents 2.5% represents about 1600 documents, still a considerable number. We selected 2.5% after observing, by manual inspection of several test queries, that this limit removed many common terms while keeping many of the most informative ones.

Ideally, the effectiveness of a term selection algorithm should scale to large sample sizes. Recall from Section 3.3 that *result set sample size* refers to the number of documents from the seed query's result set from which refinement terms are extracted. The larger the sample size, the more difficult it is to find relevant information buried in it and the greater the need for system assisted result set visualization. Larger sample sizes also

yield better coverage of the entire seed result set. Although some commercial search engines report that in practice 77% of the time users do not look beyond the first two pages of results (20 documents) [37], it is not unrealistic to think that a user may be able to manually sift a result set of 100 documents without additional system support. For this reason, configurations C and D in the lower row of Figure 3-9 use a larger result set sample of 500 documents.

### 3.4.2 Examples of Selected Terms

Tables 3.1 to 3.6 illustrate the actual terms selected by $DM_{nfx}$, *CTS* and *COMBO* under configurations C and D for three of our test queries. For each term, the table shows the change in precision (labeled $\Delta P$) and the number of documents in the result set containing the term (labeled Cov). As in Section 3.4.5, the change in precision of each term reflects the difference between the precision achieved by the seed query when augmented by the term minus the original precision of the seed query. Cov can range from 1 to 500, the document sample size. All the terms are ordered lexicographically.

These tables of example terms demonstrate the difficulty of comparing the effectiveness of various term selection algorithms by visually inspecting their output. Rather than as a means for comparing the algorithms we include the tables to give the reader an intuitive idea of the type and amount of information conveyed by the terms extracted by the different algorithms.

| $DM_{nfx}$ | | | $CTS$ | | | $COMBO$ | | |
|---|---|---|---|---|---|---|---|---|
| Suggestion | △ P | Cov | Suggestion | △ P | Cov | Suggestion | △ P | Cov |
| administration | 1 | 251 | accidents | 4 | 50 | agency | 3 | 183 |
| aeronautics | 4 | 125 | aerospace | -3 | 50 | air | 2 | 143 |
| agency | 3 | 183 | agency | 3 | 183 | america | 3 | 105 |
| air | 2 | 143 | american | 2 | 182 | american | 2 | 182 |
| astronauts | 19 | 135 | based | 4 | 176 | apollo | -1 | 49 |
| billion | -1 | 205 | beginning | 9 | 146 | atlantis | 19 | 51 |
| booster | 6 | 66 | billion | -1 | 205 | based | 4 | 176 |
| budget | 0 | 138 | broadcast | 0 | 50 | billion | -1 | 205 |
| bush | -5 | 174 | bush | -5 | 174 | building | -3 | 111 |
| center | 3 | 152 | calls | 0 | 233 | bush | -5 | 174 |
| challenged | 12 | 125 | center | 3 | 152 | calls | 0 | 233 |
| cost | 0 | 145 | community | 0 | 130 | center | 3 | 152 |
| crew | 22 | 104 | country | -2 | 131 | challenged | 12 | 125 |
| defense | -7 | 146 | day | 9 | 237 | commander | 20 | 98 |
| designer | -2 | 135 | defense | -7 | 146 | community | 0 | 130 |
| developments | 0 | 205 | developments | 0 | 205 | defense | -7 | 146 |
| earth | 15 | 162 | fields | 0 | 65 | designer | -2 | 135 |
| engineer | 4 | 119 | flight | 12 | 199 | developments | 0 | 205 |
| exploring | 9 | 94 | government | -11 | 159 | fiscal | -8 | 94 |
| fiscal | -8 | 94 | habitual | -1 | 2 | flight | 12 | 199 |
| flight | 12 | 199 | helped | -2 | 163 | future | -5 | 105 |
| land | 12 | 123 | house | -7 | 180 | helped | -2 | 163 |
| launched | 17 | 234 | included | 0 | 237 | house | -7 | 180 |
| manned | 6 | 88 | launched | 17 | 234 | included | 0 | 237 |
| mar | 7 | 78 | lives | -3 | 110 | increasing | -6 | 135 |
| missiles | -8 | 86 | missiles | -8 | 86 | launched | 17 | 234 |
| mission | 20 | 180 | mission | 20 | 180 | liftoff | 18 | 49 |
| moon | -2 | 94 | month | 12 | 211 | looking | 6 | 134 |
| nasa | 8 | 197 | moved | -3 | 120 | mission | 20 | 180 |
| orbit | 17 | 154 | nasa | 8 | 197 | money | -4 | 125 |
| payload | 9 | 63 | nearby | 0 | 16 | month | 12 | 211 |
| planet | 12 | 81 | officials | 2 | 223 | moon | -2 | 94 |
| plans | 2 | 253 | operates | 3 | 122 | nasa | 8 | 197 |
| project | 5 | 182 | pennyslvania | 0 | 1 | project | 5 | 182 |
| research | 0 | 151 | people | -1 | 204 | provide | -3 | 150 |
| rockets | 8 | 143 | project | 5 | 182 | research | 0 | 151 |
| satellite | 9 | 129 | provide | -3 | 150 | satellite | 9 | 129 |
| scheduled | 15 | 151 | publicity | -6 | 108 | science | 2 | 115 |
| science | 2 | 115 | reported | -1 | 200 | scientist | 10 | 116 |
| scientist | 10 | 116 | research | 0 | 151 | set | 3 | 161 |
| shuttle | 18 | 191 | scheduled | 15 | 151 | shuttle | 18 | 191 |
| soviet | 3 | 148 | science | 2 | 115 | soviet | 3 | 148 |
| spacecraft | 12 | 87 | scientist | 10 | 116 | starring | 7 | 113 |
| spend | 0 | 141 | shuttle | 18 | 191 | start | 9 | 133 |
| starring | 7 | 113 | system | 10 | 193 | station | 6 | 135 |
| station | 6 | 135 | technology | -1 | 128 | studies | 6 | 101 |
| system | 10 | 193 | trip | 6 | 50 | system | 10 | 193 |
| technology | -1 | 128 | united | 0 | 174 | technology | -1 | 128 |
| testing | -2 | 133 | voyaging | 7 | 50 | testing | -2 | 133 |
| unmanned | 16 | 71 | week | -3 | 169 | voyaging | 7 | 50 |

Table 3.1: Terms generated by algorithms *CTS*, $DM_{nfx}$ and *COMBO* for TREC topic number 10. Seed query = `Space Program`. Initial Precision = 12%. Document frequency limit = 100%. Result set sample size = 500.

| $DM_{nfx}$ | | | $CTS$ | | | $COMBO$ | | |
|---|---|---|---|---|---|---|---|---|
| Suggestion | △ P | Cov | Suggestion | △ P | Cov | Suggestion | △ P | Cov |
| aeronautics | 4 | 125 | aeronautics | 4 | 125 | accomplishments | 5 | 29 |
| aerospace | -3 | 50 | aerospace | -3 | 50 | aeronautics | 4 | 125 |
| apollo | -1 | 49 | afford | -3 | 27 | aerospace | -3 | 50 |
| astronauts | 19 | 135 | apollo | -1 | 49 | apollo | -1 | 49 |
| atlantis | 19 | 51 | applicants | 0 | 29 | astronauts | 19 | 135 |
| atmosphere | 8 | 66 | audiences | -3 | 25 | atmosphere | 8 | 66 |
| booster | 6 | 66 | balance | -1 | 22 | capability | 2 | 57 |
| canaveral | 6 | 30 | beautiful | -1 | 12 | columbia | 6 | 34 |
| capability | 2 | 57 | bed | 0 | 16 | complex | 2 | 34 |
| cosmonaut | 6 | 36 | branch | -1 | 9 | contractors | -3 | 30 |
| craft | 8 | 41 | breath | 0 | 11 | cosmonaut | 6 | 36 |
| dan | -1 | 51 | buyers | -2 | 15 | craft | 8 | 41 |
| discoveries | 6 | 53 | cable | -2 | 22 | dan | -1 | 51 |
| earth | 15 | 162 | capitol | -2 | 28 | disaster | 5 | 37 |
| exploring | 9 | 94 | catalog | -1 | 3 | earth | 15 | 162 |
| explosion | 6 | 68 | categories | -2 | 16 | era | 1 | 35 |
| galileo | 12 | 30 | coordinating | -2 | 18 | exploring | 9 | 94 |
| jupiter | 11 | 39 | decrease | -1 | 9 | explosion | 6 | 68 |
| laboratory | 6 | 70 | earth | 15 | 162 | flew | 2 | 36 |
| liftoff | 18 | 49 | ehrenhalt | 0 | 1 | houston | 4 | 37 |
| lunar | -2 | 31 | electronic | 0 | 25 | image | 9 | 44 |
| magellan | 16 | 30 | enroll | -1 | 6 | kennedy | 5 | 45 |
| manned | 6 | 88 | explosion | 6 | 68 | laboratory | 6 | 70 |
| mar | 7 | 78 | extra | 0 | 31 | laser | -2 | 22 |
| mir | 7 | 31 | homeless | -3 | 24 | manned | 6 | 88 |
| missiles | -8 | 86 | looting | -1 | 2 | mapped | 15 | 35 |
| moon | -2 | 94 | makers | -2 | 13 | mar | 7 | 78 |
| nasa | 8 | 197 | manned | 6 | 88 | mir | 7 | 31 |
| orbit | 17 | 154 | martinez | -1 | 2 | missiles | -8 | 86 |
| pad | 9 | 52 | medicare | -3 | 22 | nasa | 8 | 197 |
| payload | 9 | 63 | missiles | -8 | 86 | ocean | 7 | 44 |
| pentagon | -10 | 68 | nasa | 8 | 197 | pentagon | -10 | 68 |
| planet | 12 | 81 | offset | -1 | 9 | permanently | 1 | 47 |
| planetary | 16 | 35 | ohrenstein | 0 | 1 | photographs | 9 | 37 |
| probe | 16 | 75 | paying | -2 | 19 | pioneered | 10 | 32 |
| propulsion | 5 | 30 | satellite | 9 | 129 | planet | 12 | 81 |
| rockets | 8 | 143 | scotland | -1 | 5 | priority | -2 | 45 |
| satellite | 9 | 129 | shortage | 0 | 17 | radiation | 2 | 25 |
| scientific | 6 | 64 | shuttle | 18 | 191 | satellite | 9 | 129 |
| shuttle | 18 | 191 | solution | -2 | 19 | scientific | 6 | 64 |
| solar | 15 | 55 | speed | 1 | 28 | shield | -1 | 27 |
| spacecraft | 12 | 87 | strategic | -8 | 60 | shuttle | 18 | 191 |
| specialists | 17 | 44 | tasks | 1 | 29 | solid | 2 | 35 |
| strategic | -8 | 60 | thirdly | 0 | 1 | specialists | 17 | 44 |
| surface | 16 | 62 | veterans | 1 | 54 | strategic | -8 | 60 |
| truly | 4 | 38 | voyaging | 7 | 50 | tenn | -2 | 29 |
| unmanned | 16 | 71 | wastes | -2 | 29 | trillion | -2 | 27 |
| venue | 17 | 36 | welfare | -2 | 8 | truly | 4 | 38 |
| veterans | 1 | 54 | wheel | -1 | 10 | veterans | 1 | 54 |
| voyaging | 7 | 50 | wuld | 0 | 1 | voyaging | 7 | 50 |

Table 3.2: Terms generated by algorithms $DM_{nfx}$, *CTS* and *COMBO* for TREC topic number 10. Seed query = `Space Program`. Initial Precision = 12%. Document frequency limit = 2.5%. Result set sample size = 500.

| $DM_{nfx}$ | | | $CTS$ | | | $COMBO$ | | |
|---|---|---|---|---|---|---|---|---|
| Suggestion | Δ P | Cov | Suggestion | Δ P | Cov | Suggestion | Δ P | Cov |
| administration | 1 | 144 | advanced | 0 | 123 | according | 0 | 119 |
| advanced | 0 | 123 | american | 0 | 226 | advanced | 0 | 123 |
| american | 0 | 226 | based | 1 | 158 | aid | 2 | 91 |
| associated | -1 | 171 | blocked | 2 | 50 | american | 0 | 226 |
| based | 1 | 158 | calls | 1 | 188 | based | 1 | 158 |
| blood | 2 | 73 | care | -1 | 157 | business | -1 | 124 |
| care | -1 | 157 | center | 2 | 179 | calls | 1 | 188 |
| center | 2 | 179 | charges | -1 | 82 | care | -1 | 157 |
| clinic | 3 | 69 | charity | 0 | 4 | center | 2 | 179 |
| company | 0 | 167 | city | -1 | 96 | community | 0 | 105 |
| computer | 0 | 86 | company | 0 | 167 | company | 0 | 167 |
| controls | -1 | 121 | depends | -1 | 50 | department | 0 | 144 |
| cost | 0 | 130 | developments | 0 | 195 | developments | 0 | 195 |
| developments | 0 | 195 | directors | 1 | 160 | directors | 1 | 160 |
| directors | 1 | 160 | doctoral | 2 | 165 | doctoral | 2 | 165 |
| disease | 3 | 91 | ec | 0 | 1 | dr | 3 | 137 |
| doctoral | 2 | 165 | fort | 0 | 21 | equipment | -1 | 79 |
| dr | 3 | 137 | government | -1 | 183 | found | 0 | 129 |
| effect | 2 | 133 | health | -1 | 190 | fund | -1 | 101 |
| electronic | -1 | 64 | hospital | 2 | 169 | health | -1 | 190 |
| health | -1 | 190 | included | 0 | 243 | helped | 1 | 199 |
| helped | 1 | 199 | issue | -1 | 149 | hospital | 2 | 169 |
| hospital | 2 | 169 | learn | 0 | 50 | human | -1 | 96 |
| human | -1 | 96 | market | -1 | 104 | included | 0 | 243 |
| included | 0 | 243 | military | 0 | 78 | industry | 1 | 149 |
| industry | 1 | 149 | million | -1 | 196 | life | -1 | 128 |
| institution | 1 | 135 | officers | -1 | 147 | major | 1 | 136 |
| laboratory | 1 | 63 | officials | -1 | 168 | million | -1 | 196 |
| life | -1 | 128 | people | 0 | 201 | nurse | -1 | 52 |
| medicine | 1 | 113 | percent | 1 | 178 | plans | 0 | 153 |
| nationally | -1 | 273 | procedure | 2 | 61 | procedure | 2 | 61 |
| patient | 4 | 155 | questioned | -1 | 94 | products | 0 | 126 |
| percent | 1 | 178 | reaction | 0 | 15 | program | -1 | 147 |
| physician | -1 | 97 | rep | -1 | 50 | received | -1 | 115 |
| products | 0 | 126 | reported | 1 | 220 | recommendation | -1 | 76 |
| program | -1 | 147 | research | 2 | 208 | research | 2 | 208 |
| project | -1 | 109 | resulted | 4 | 109 | robert | 0 | 91 |
| provide | -1 | 161 | rule | -1 | 74 | school | 0 | 128 |
| requiring | 0 | 118 | savings | 1 | 50 | science | -1 | 102 |
| research | 2 | 208 | saying | -1 | 68 | seek | 0 | 100 |
| school | 0 | 128 | service | -1 | 144 | service | -1 | 144 |
| science | -1 | 102 | system | -1 | 168 | studies | 1 | 157 |
| scientist | 1 | 70 | testing | 2 | 117 | surgery | 3 | 52 |
| studies | 1 | 157 | time | -1 | 229 | time | -1 | 229 |
| system | -1 | 168 | united | 0 | 190 | transferred | -1 | 60 |
| testing | 2 | 117 | university | 1 | 189 | treated | 2 | 88 |
| treated | 2 | 88 | utmb | 0 | 1 | treatment | 4 | 101 |
| treatment | 4 | 101 | wednesday | 0 | 84 | united | 0 | 190 |
| united | 0 | 190 | west | -1 | 72 | university | 1 | 189 |
| university | 1 | 189 | white | 0 | 78 | using | 3 | 84 |

Table 3.3: Terms generated by algorithms *CTS*, $DM_{nfx}$ and *COMBO* for TREC topic number 23. Seed query = `Medical Technology`. Initial Precision 1%. Document frequency limit = 100%. Result set sample size = 500.

| $DM_{nfx}$ | | | $CTS$ | | | $COMBO$ | | |
|---|---|---|---|---|---|---|---|---|
| Suggestion | △ P | Cov | Suggestion | △ P | Cov | Suggestion | △ P | Cov |
| applicants | 0 | 46 | aborted | -1 | 31 | aborted | -1 | 31 |
| birth | -1 | 33 | academy | 0 | 26 | academy | 0 | 26 |
| brain | 2 | 43 | ann | 1 | 23 | aerospace | 0 | 23 |
| cancer | 7 | 51 | brain | 2 | 43 | applicants | 0 | 46 |
| capability | -1 | 35 | brunswick | 0 | 3 | aren | 0 | 27 |
| cell | 7 | 30 | cancer | 7 | 51 | brain | 2 | 43 |
| clinic | 3 | 69 | chamber | 0 | 9 | cancer | 7 | 51 |
| competing | -1 | 49 | clinic | 3 | 69 | capability | -1 | 35 |
| device | 1 | 36 | competing | -1 | 49 | chips | -1 | 31 |
| diabetes | 2 | 28 | desire | 0 | 17 | clinic | 3 | 69 |
| die | 1 | 36 | device | 1 | 36 | competing | -1 | 49 |
| electronic | -1 | 64 | dividend | 0 | 5 | complex | 0 | 31 |
| ethics | -1 | 36 | ducatman | 0 | 1 | cure | 2 | 22 |
| fighter | -1 | 33 | ec | 0 | 1 | device | 1 | 36 |
| fsx | 0 | 22 | editions | 0 | 17 | diagnose | 1 | 19 |
| genetically | 3 | 23 | electronic | -1 | 64 | donated | -1 | 26 |
| graduate | 0 | 34 | fighter | -1 | 33 | electronic | -1 | 64 |
| harvard | -1 | 37 | flavor | 0 | 4 | evaluating | 0 | 25 |
| hdtv | 0 | 17 | flow | 1 | 25 | faculty | 0 | 19 |
| infant | 1 | 30 | geneva | 0 | 8 | fighter | -1 | 33 |
| infection | 3 | 38 | gradually | 0 | 15 | foundation | 1 | 27 |
| journals | 2 | 46 | innovation | 0 | 17 | journals | 2 | 46 |
| lab | 0 | 25 | integration | 0 | 19 | knowledge | 0 | 26 |
| laboratory | 1 | 63 | keizai | 0 | 1 | laboratory | 1 | 63 |
| licenses | -1 | 52 | laboratory | 1 | 63 | licenses | -1 | 52 |
| massachusetts | -1 | 46 | licenses | -1 | 52 | lung | 2 | 24 |
| medicine | 1 | 113 | lies | 0 | 15 | makers | 0 | 28 |
| methods | 2 | 32 | makers | 0 | 28 | massachusetts | -1 | 46 |
| needles | 2 | 25 | massachusetts | -1 | 46 | medicine | 1 | 113 |
| nurse | -1 | 52 | measles | 0 | 3 | methods | 2 | 32 |
| pain | 1 | 43 | medicine | 1 | 113 | nurse | -1 | 52 |
| patient | 4 | 155 | multi | 0 | 6 | pain | 1 | 43 |
| physical | -1 | 35 | patient | 4 | 155 | patient | 4 | 155 |
| physician | -1 | 97 | patrol | 0 | 5 | permanently | 0 | 27 |
| professional | -1 | 35 | physician | -1 | 97 | physician | -1 | 97 |
| ray | 0 | 31 | pompidou | 0 | 1 | scientific | -1 | 48 |
| scientific | -1 | 48 | professional | -1 | 35 | sensitive | 0 | 33 |
| screen | 1 | 35 | reaction | 0 | 15 | sick | 1 | 27 |
| sensitive | 0 | 33 | replied | 0 | 6 | specialists | -1 | 41 |
| specialists | -1 | 41 | retirees | 0 | 4 | speed | 0 | 30 |
| subcommittee | -1 | 34 | rural | 0 | 15 | subcommittee | -1 | 34 |
| surgeon | -1 | 34 | shareholders | 0 | 10 | surgery | 3 | 52 |
| surgery | 3 | 52 | structures | 2 | 23 | syndrome | 1 | 20 |
| surgical | 0 | 29 | surgery | 3 | 52 | teaches | 0 | 30 |
| tech | 0 | 31 | technological | 0 | 34 | tech | 0 | 31 |
| technological | 0 | 34 | utmb | 0 | 1 | technological | 0 | 34 |
| therapy | 8 | 27 | venture | -1 | 35 | trends | 0 | 28 |
| tools | 1 | 30 | veterans | 0 | 33 | venture | -1 | 35 |
| venture | -1 | 35 | wastes | -1 | 32 | veterans | 0 | 33 |
| wastes | -1 | 32 | wynder | 0 | 1 | wastes | -1 | 32 |

Table 3.4: Terms generated by algorithms *CTS*, $DM_{nfx}$ and *COMBO* for TREC topic number 23. Seed query = `Medical Technology`. Initial Precision 1%. Document frequency limit = 2.5%. Result set sample size = 500.

| $DM_{nfx}$ | | | $CTS$ | | | $COMBO$ | | |
|---|---|---|---|---|---|---|---|---|
| Suggestion | △ P | Cov | Suggestion | △ P | Cov | Suggestion | △ P | Cov |
| activists | 6 | 159 | announced | 4 | 104 | activists | 6 | 159 |
| africa | -4 | 465 | arm | -2 | 83 | adriaan | -9 | 47 |
| anc | 34 | 124 | calls | 2 | 240 | affair | 2 | 125 |
| anglican | 2 | 71 | condemnations | -3 | 50 | ban | 17 | 141 |
| angola | -6 | 102 | day | -3 | 196 | campaign | 8 | 166 |
| anti | 14 | 261 | de | 5 | 232 | de | 5 | 232 |
| apartheid | 10 | 334 | difficult | -7 | 34 | durban | 0 | 55 |
| ban | 17 | 141 | effort | -4 | 84 | elected | 7 | 226 |
| botha | 9 | 148 | elected | 7 | 226 | fighting | 4 | 141 |
| campaign | 8 | 166 | guerrilla | 8 | 246 | force | -3 | 195 |
| cape | 15 | 131 | incident | -6 | 50 | foreign | -1 | 159 |
| congress | 16 | 268 | included | 1 | 245 | free | 15 | 143 |
| country | 4 | 318 | independence | -8 | 177 | guerrilla | 8 | 246 |
| de | 5 | 232 | klerk | 7 | 194 | independence | -8 | 177 |
| desmond | 1 | 70 | largest | 7 | 135 | johannesburg | 9 | 152 |
| elected | 7 | 226 | meeting | 7 | 206 | klerk | 7 | 194 |
| freed | 24 | 110 | million | 2 | 236 | largest | 7 | 135 |
| guerrilla | 8 | 246 | ministers | 1 | 207 | led | 4 | 128 |
| independence | -8 | 177 | month | 10 | 188 | maintains | 5 | 102 |
| johannesburg | 9 | 152 | multiracial | -10 | 45 | mandela | 29 | 133 |
| klerk | 7 | 194 | nelson | 25 | 132 | meeting | 7 | 206 |
| leader | 18 | 351 | officers | -1 | 135 | ministers | 1 | 207 |
| major | 3 | 274 | plans | 1 | 193 | movement | 15 | 188 |
| mandela | 29 | 133 | police | 3 | 220 | mozambique | -5 | 48 |
| ministers | 1 | 207 | powerful | 2 | 167 | namibia | -15 | 122 |
| movement | 15 | 188 | recently | 6 | 169 | nationalists | 3 | 95 |
| namibia | -15 | 122 | related | 0 | 88 | negotiation | 20 | 175 |
| nationalists | 3 | 95 | released | 17 | 197 | nelson | 25 | 132 |
| nationally | 7 | 436 | reported | -7 | 229 | opposed | 7 | 105 |
| negotiation | 20 | 175 | requiring | -8 | 47 | parties | 4 | 200 |
| nelson | 25 | 132 | restriction | 1 | 90 | peace | 11 | 174 |
| organized | 7 | 273 | rule | 0 | 185 | pik | -6 | 48 |
| outlawed | 22 | 121 | saying | 3 | 143 | police | 3 | 220 |
| peace | 11 | 174 | school | -2 | 123 | policy | 4 | 166 |
| police | 3 | 220 | secretary | -2 | 131 | pretoria | -2 | 126 |
| politics | 14 | 275 | shapi | 0 | 2 | protest | 1 | 134 |
| pretoria | -2 | 126 | slutsky | -1 | 1 | race | 0 | 142 |
| prisoner | 28 | 151 | stage | -4 | 50 | racially | 5 | 164 |
| race | 0 | 142 | statement | 3 | 145 | recently | 6 | 169 |
| racially | 5 | 164 | supporters | 8 | 217 | released | 17 | 197 |
| sabotage | 22 | 85 | time | 1 | 192 | rule | 0 | 185 |
| sanctions | 4 | 95 | town | 9 | 166 | sabotage | 22 | 85 |
| segregated | 1 | 141 | united | -7 | 235 | sanctions | 4 | 95 |
| sisulu | 15 | 57 | violating | -10 | 50 | segregated | 1 | 141 |
| town | 9 | 166 | wake | -2 | 2 | separate | 1 | 129 |
| township | 9 | 116 | walter | 11 | 50 | supporters | 8 | 217 |
| tutu | 0 | 69 | week | 0 | 229 | territory | -16 | 103 |
| vote | 9 | 172 | world | -5 | 119 | united | -7 | 235 |
| white | 5 | 370 | worlock | 0 | 1 | vlok | -10 | 47 |
| zambia | 11 | 76 | zambia | 11 | 76 | zimbabwe | -2 | 48 |

Table 3.5: Terms generated by algorithms *CTS*, $DM_{nfx}$ and *COMBO* for TREC topic number 109. Seed query = `Black Resistance Against the South African Government`. Initial Precision 32%. Document frequency limit = 100 %. Result set sample size = 500.

| $DM_{nfx}$ | | | $CTS$ | | | $COMBO$ | | |
|---|---|---|---|---|---|---|---|---|
| Suggestion | △ P | Cov | Suggestion | △ P | Cov | Suggestion | △ P | Cov |
| adriaan | -9 | 47 | acronym | -1 | 8 | afrikaans | -8 | 38 |
| afrikaans | -8 | 38 | adenkule | -1 | 1 | alliance | -4 | 45 |
| afrikaner | 0 | 33 | adr | -1 | 2 | anc | 34 | 124 |
| anc | 34 | 124 | affirmatively | -2 | 2 | angola | -6 | 102 |
| anglican | 2 | 71 | anc | 34 | 124 | archbishop | 0 | 71 |
| angola | -6 | 102 | angola | -6 | 102 | botha | 9 | 148 |
| apartheid | 10 | 334 | aside | 0 | 6 | boycott | -3 | 54 |
| archbishop | 0 | 71 | assassinated | -10 | 26 | cape | 15 | 131 |
| boesak | -1 | 35 | bonn | -2 | 6 | clashes | -6 | 44 |
| botha | 9 | 148 | botha | 9 | 148 | colleagues | 6 | 54 |
| boycott | -3 | 54 | cape | 15 | 131 | colonial | -12 | 53 |
| cape | 15 | 131 | clashes | -6 | 44 | condemnations | -3 | 50 |
| cuban | -16 | 58 | condemnations | -3 | 50 | cuba | -10 | 51 |
| defiance | -6 | 46 | continents | -5 | 23 | desmond | 1 | 70 |
| desmond | 1 | 70 | cuba | -10 | 51 | detained | -4 | 52 |
| durban | 0 | 55 | detained | -4 | 52 | detentions | -8 | 36 |
| exile | 9 | 64 | excutive | -1 | 1 | discriminated | 2 | 58 |
| freed | 24 | 110 | fred | -7 | 17 | durban | 0 | 55 |
| imprisoned | 17 | 78 | goodyear | -1 | 2 | exile | 9 | 64 |
| johannesburg | 9 | 152 | homeland | 0 | 45 | freed | 24 | 110 |
| klerk | 7 | 194 | indian | 0 | 62 | homeland | 0 | 45 |
| lusaka | 11 | 37 | johannesburg | 9 | 152 | indian | 0 | 62 |
| mandela | 29 | 133 | klerk | 7 | 194 | johannesburg | 9 | 152 |
| militant | 8 | 62 | locked | 1 | 8 | klerk | 7 | 194 |
| mozambique | -5 | 48 | mainly | -6 | 23 | mandela | 29 | 133 |
| multiracial | -10 | 45 | mandela | 29 | 133 | marxist | -12 | 56 |
| namibia | -15 | 122 | mobotu | -1 | 1 | mozambique | -5 | 48 |
| namibian | -18 | 61 | mount | -5 | 25 | namibia | -15 | 122 |
| nationalists | 3 | 95 | mozambique | -5 | 48 | nationalists | 3 | 95 |
| nelson | 25 | 132 | namibia | -15 | 122 | neighborhood | -3 | 51 |
| outlawed | 22 | 121 | nelson | 25 | 132 | nelson | 25 | 132 |
| parliamentary | 2 | 75 | outlawed | 22 | 121 | outlawed | 22 | 121 |
| pik | -6 | 48 | palestinian | -2 | 5 | policemen | -6 | 43 |
| plotted | 21 | 61 | permission | -2 | 35 | pretoria | -2 | 126 |
| pretoria | -2 | 126 | pretoria | -2 | 126 | racially | 5 | 164 |
| prominent | 6 | 73 | prize | -4 | 22 | racist | -10 | 31 |
| racially | 5 | 164 | racially | 5 | 164 | repealing | -4 | 35 |
| rev | 5 | 64 | rev | 5 | 64 | rev | 5 | 64 |
| sabotage | 22 | 85 | sabotage | 22 | 85 | sabotage | 22 | 85 |
| sanctions | 4 | 95 | sanctions | 4 | 95 | sanctions | 4 | 95 |
| segregated | 1 | 141 | seal | -1 | 8 | segregated | 1 | 141 |
| sisulu | 15 | 57 | segregated | 1 | 141 | supervised | -13 | 44 |
| soweto | 6 | 57 | stagnant | -2 | 2 | thatcher | -1 | 43 |
| swapo | -17 | 58 | thatcher | -1 | 43 | township | 9 | 116 |
| township | 9 | 116 | township | 9 | 116 | transition | -15 | 61 |
| transition | -15 | 61 | transition | -15 | 61 | tutu | 0 | 69 |
| tutu | 0 | 69 | unheeded | -1 | 1 | van | -5 | 43 |
| vlok | -10 | 47 | untested | -1 | 2 | vetoed | 3 | 38 |
| zambia | 11 | 76 | vetoed | 3 | 38 | walter | 11 | 50 |
| zimbabwe | -2 | 48 | walter | 11 | 50 | zimbabwe | -2 | 48 |

Table 3.6: Terms generated by algorithms *CTS*, $DM_{nfx}$ and *COMBO* for TREC topic number 109. Seed query = `Black Resistance Against the South African Government`. Initial Precision 32%. Document frequency limit = 2.5%. Result set sample size = 500.

### 3.4.3 Partition Difference

This section measures the redundancy and information loss achieved by each term selection algorithm under our four experimental configurations. In the experiment, each algorithm is required to generate 50 query refinement terms for each of 150 short test queries.

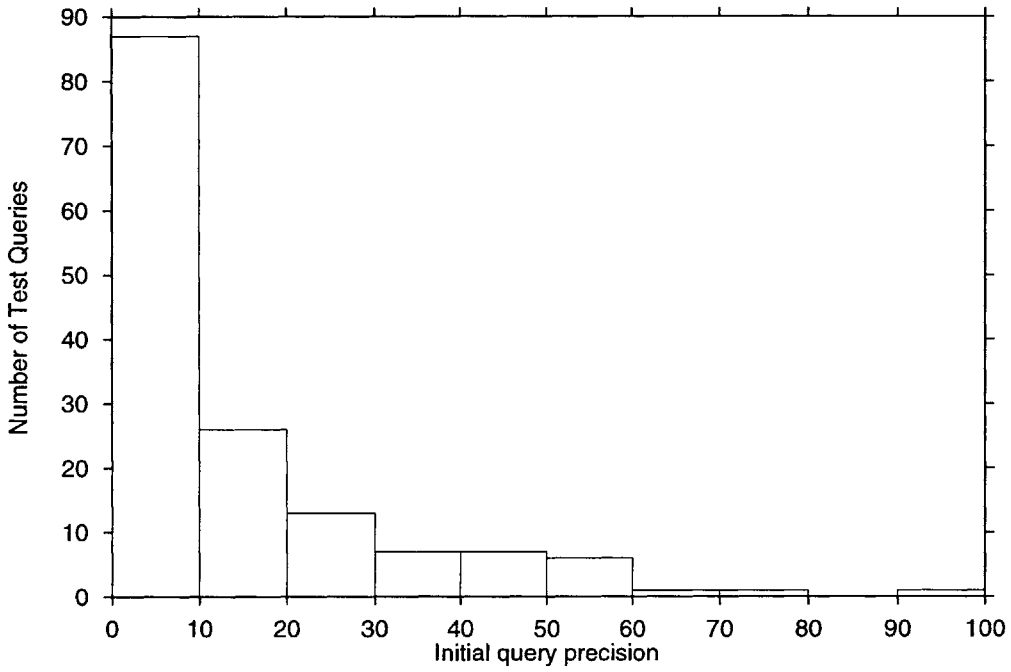



Figure 3-10: Distribution of test queries by initial query precision. 84K documents, 150 queries

As in Chapter 2, we will measure the redundancy and information loss of terms considering the initial precision of the seed query. Figure 3-10 shows the distribution of our 150 test queries by initial precision. The number of queries achieving more than 60% precision is so small that we will not consider any results for this range of initial precision any further.

Figure 3-11 shows one plot for each of our four experimental configurations. The plots are laid out in the same way as the configurations in Figure 3-9. Configurations A and B

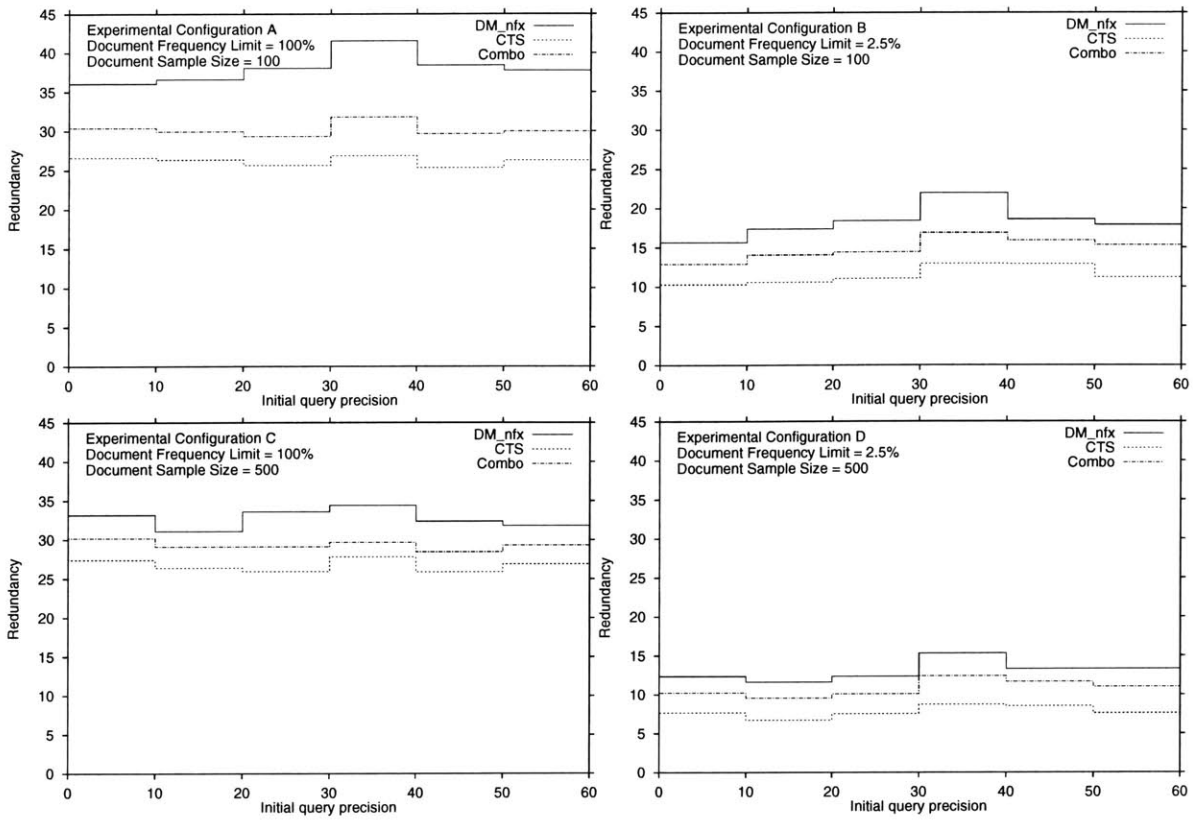



Figure 3-11: Average proportion of covering refinement terms per result set document ignoring documents not covered by any term.

appear in the upper row, while C and D appear in the lower row. Configurations A and C make up the left column while B and D make up the right one. The horizontal axis indicates test query precision using a discrete scale with buckets representing intervals 10 percentage points wide. For instance, the first interval groups queries with precision in the semi-open interval $[0\%, 10\%)$.

The vertical axis measures term redundancy as the average proportion of covering refinement terms per result set document. A term is said to cover a document if it appears at least once in the document. In order to isolate the measurement of redundancy from that of information loss, the documents not covered by any selected terms are ignored in computing the averages. For example, the plot for configuration C in Figure 3-11 shows

that on average, in the $[0, 1)$ initial precision interval, each document in the result set sample contains 27.4% of the terms selected by *CTS*.

The most important conclusion we can draw from Figure 3-11 is that redundancy appears to be a problem in practice. Under configuration A, $DM_{nfx}$ achieves redundancy ranging from 35% to 45%. That means that out of the 50 terms selected by the algorithm, 17 to 22 terms appear in the average document. This can only be possible if there is significant overlap among the subsets of documents described by different terms.

It can also be seen in Figure 3-11 that redundancy increases with document frequency. This result confirms our expectations. Selecting terms with higher document frequencies should yield higher redundancy simply because these terms appear in a larger fraction of the result set. Figure 3-11 also illustrates the effect of combining $DM_{nfx}$ with *CTS* term selection on redundancy. Under all four configurations *COMBO* achieves redundancy that is always in between those of $DM_{nfx}$ and *CTS*.

Although some interesting observations can be drawn from the graphs in Figure 3-11 alone, it is crucial that we examine the issue of information loss before we draw any conclusions about the relative effectiveness of the various term selection algorithms. As in Figure 3-11, Figure 3-12 includes one graph for each of the four experimental configurations. Each graph plots the average proportion of result set documents not covered by any selected term. This time, the documents that are covered by at least one selected term are ignored. For instance, under configuration D and for the 0-10% initial precision interval $DM_{nfx}$ achieves an average information loss of 10.2%. This means that on average 50 out of the 500 documents in the result set are not covered by any of the terms selected by $DM_{nfx}$. This result shows how an algorithm not having information coverage as a goal may leave a significant portion of a result set uncovered by the set of selected terms.

Combining the results of Figures 3-11 and 3-12 we can see that information loss can always be reduced at the expense of increased redundancy. When no document frequency filter is used, information loss drops virtually to zero as shown by the two graphs on the

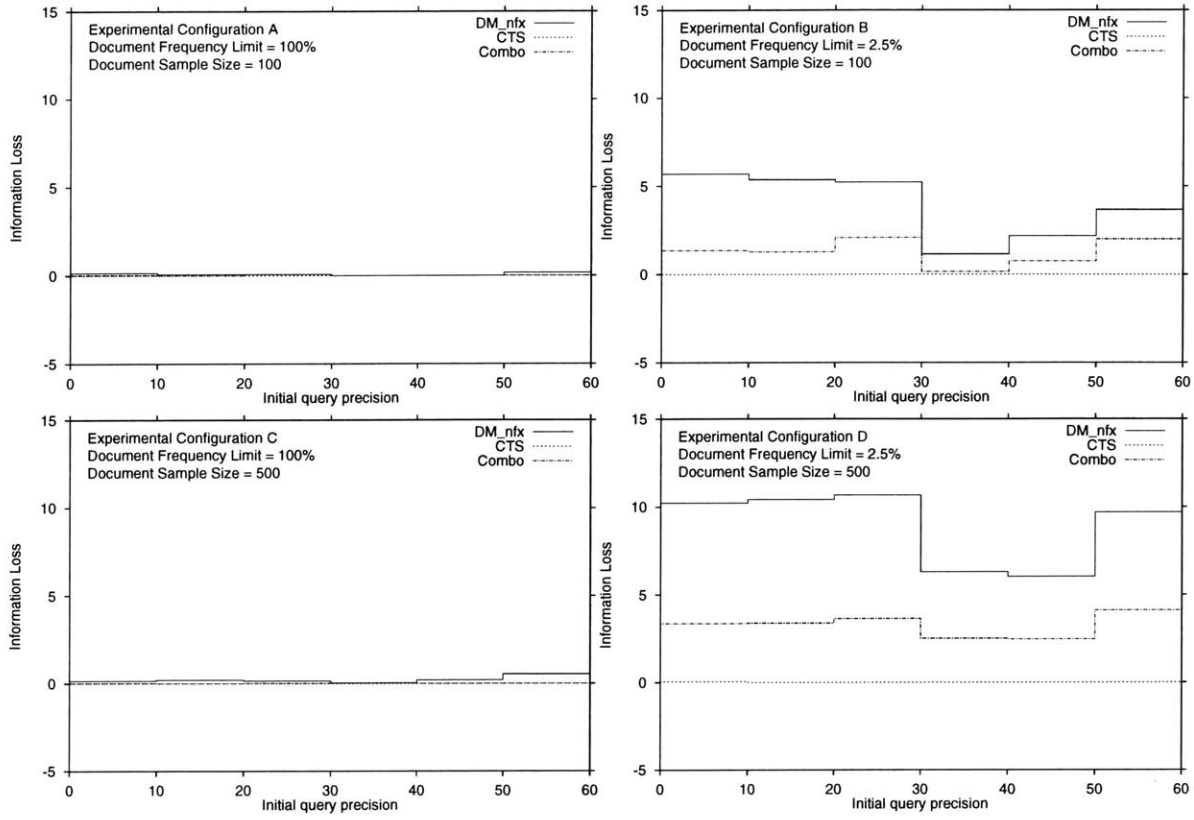



Figure 3-12: Average number of uncovered documents per query achieved by $DM_{nfx}$, *CTS* and *COMBO*.

left column of Figure 3-12. However, the corresponding redundancy graphs in Figure 3-11 show that all three algorithms yield much higher redundancy when no document filter is applied. Our goal is to find an algorithm capable of selecting terms that yield both low redundancy and low information loss.

Figures 3-11 and 3-12 also show that, under all four configurations, *CTS*, our new coverage term selection algorithm, achieves both lower redundancy and lower information loss than $DM_{nfx}$. This can be more easily seen if we combine redundancy and information loss into the single notion of partition difference that we defined in Section 3.2. Figure 3-13 illustrates the partition difference achieved by the various algorithms under our four experimental configurations. The horizontal axis represents initial query precision. The

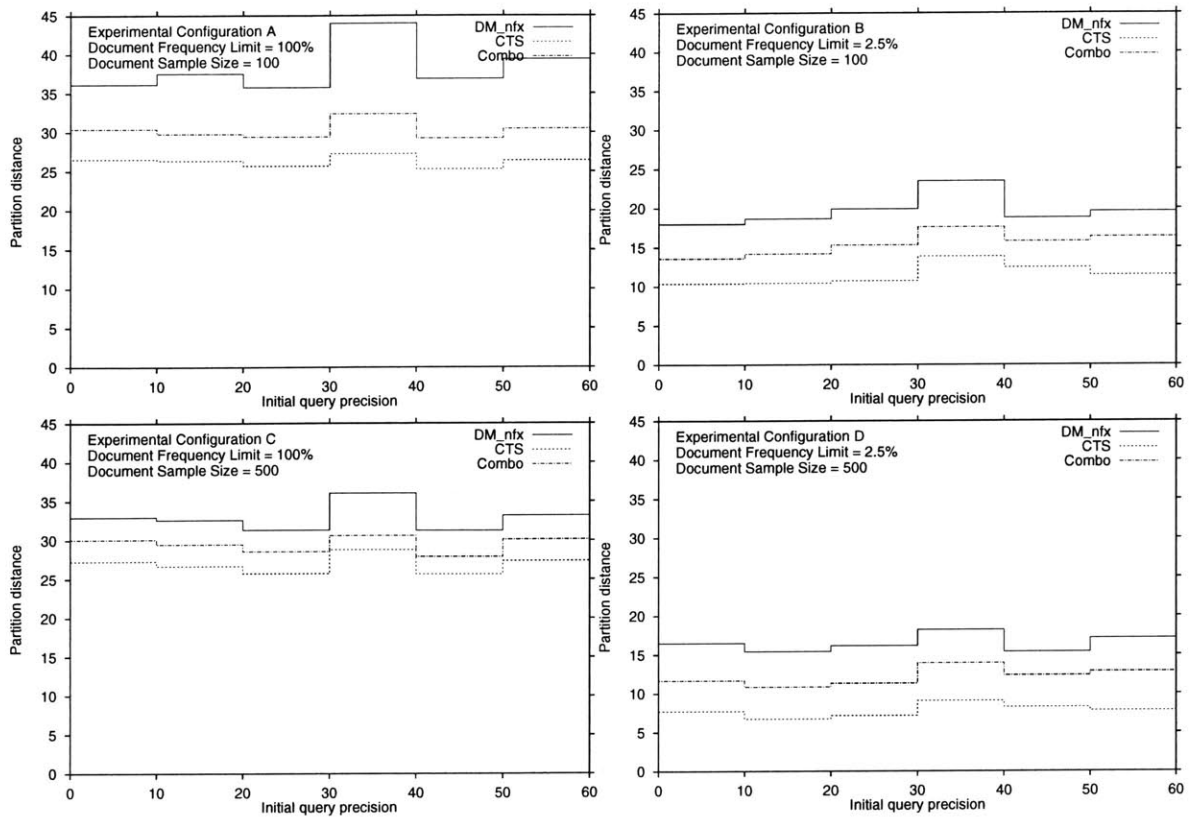



Figure 3-13: Partition difference achieved by $DM_{nfx}$, *CTS*, and *COMBO* ($loss - penalty = 0.5$).

vertical axis represents average partition difference with a $loss - penalty$ of 0.5. The results in Figure 3-13 demonstrate the superiority of *CTS* over $DM_{nfx}$ at reducing both redundancy and information loss across the entire precision range. Again, *COMBO* achieves lower partition difference than $DM_{nfx}$ but higher than *CTS*.

Although the experiments reported in this section demonstrate the superiority of *CTS* and *COMBO* in dealing with redundancy and information loss it is important that we examine the performance of these algorithms under the metrics developed in Chapter 2. The following sections present experiments comparing our experimental term selection algorithms in terms of concept recall and precision improvement.

### 3.4.4 Concept Recall

*Concept recall* measures the ability of an algorithm to recommend terms that are semantically related to the user's information need. A complete description of this experimental metric was introduced in Section 2.4.2. Each topic (test query) in the TREC collection has an associated set of concepts. These concepts were selected by the authors of the test queries to summarize the information need encoded by the queries. Concept recall measures, for each test query, what fraction of its associated concepts are selected by a query refinement algorithm as term suggestions.

Attempting to minimize redundancy and information loss may have a negative impact on concept recall. To understand why, it is useful to go back to the notion of *search paths* that we introduced in Section 3.2. A search path is a subarea of interest that a user may follow on his/her way to focusing a more general seed query. A traditional frequency-based term selection algorithm like $DM_{nfx}$ is not constrained to minimize redundant terms and information loss like *CTS*. To achieve these objectives, *CTS* maximizes the proportion of the available search paths represented by the selected terms by maximizing the proportion of documents covered by these terms.

Concept recall measures the proportion of concepts associated with a topic that an algorithm is able to select given an under-specified seed query. All search paths are not equally likely to match the concepts associated with a topic. Consider the hypothetical query `life threatening disease`. It is more likely that a topic in a test collection focuses on a well-known disease like `cancer` or `AIDS` than on more rare diseases like `legionnaire's disease`. Not considering the relative likelihood of the available search paths, *CTS* will try to select terms representing the less likely search paths at the expense of redundant terms representing more likely paths. As a consequence, concept recall will decrease.

It is unclear to us whether an algorithm that achieves much higher concept recall than another is more effective. On one extreme, an algorithm that achieves perfect (100%) concept recall is likely not to be effective, because it will only help users that happen to

be interested in the same search path that the algorithm predicts. At the other extreme, an algorithm that achieves no concept recall is even less useful. Such an algorithm never predicts a relevant search path and therefore is unlikely to help any user at all. Optimal concept recall lies somewhere between these extremes. In this section we show that the negative impact of *CTS* on concept recall is not as significant as to render the algorithm significantly less effective than $DM_{nfx}$.

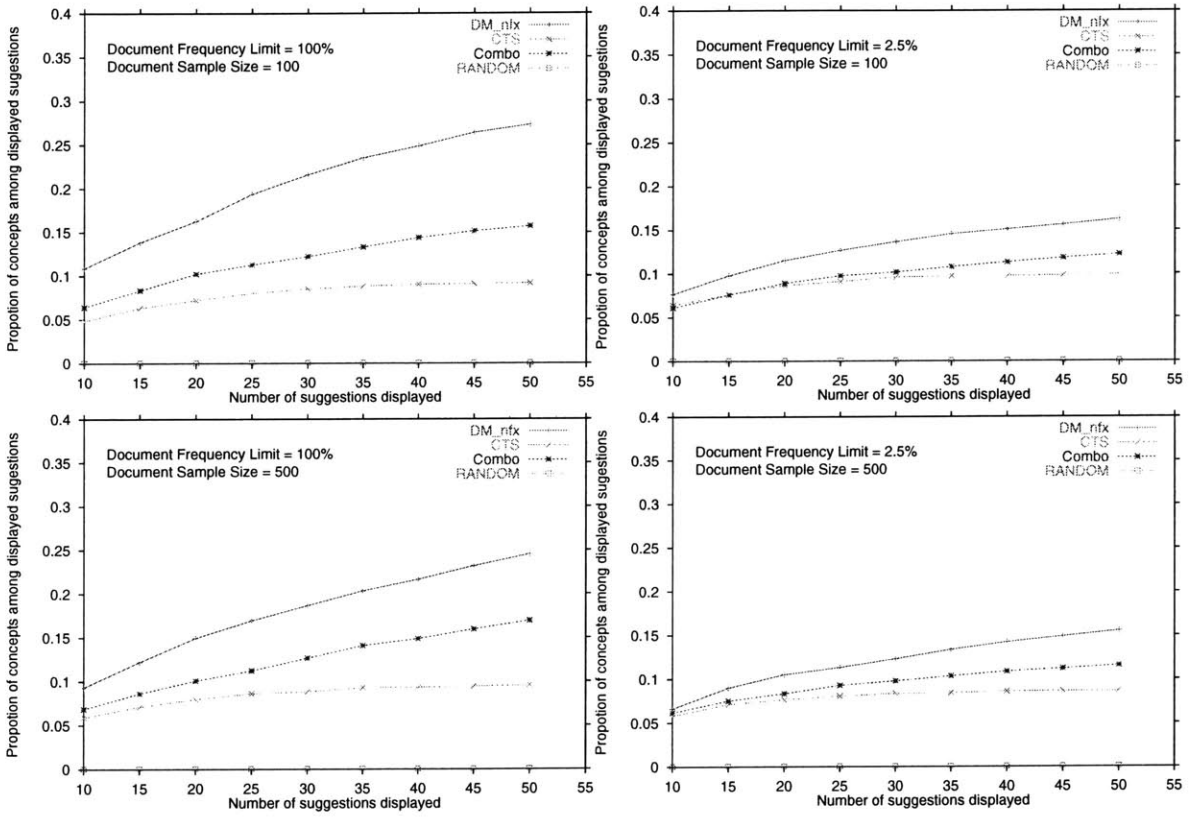



Figure 3-14: Concept Recall Achieved by *CTS*, $DM_{nfx}$ and *RANDOM* algorithms. 84K document, 150 test queries

Figure 3-14 shows concept recall graphs for each of our four experimental configurations. Each graph includes results for the three experimental algorithms ($DM_{nfx}$, *CTS* and *COMBO*) plus a control algorithm (*RANDOM*) that randomly selects terms from documents in the seed query's result set. The vertical axis shows the average proportion,

across 150 test queries, of concepts appearing among the selected terms. The horizontal axis represents the total number of terms selected. As should be expected, the proportion of concepts selected increases with the number of terms selected for all the algorithms except *RANDOM*.

Experimental configuration *A* (upper left) shows *CTS* achieving substantially less concept recall than $DM_{nfx}$. This difference is caused by $DM_{nfx}$ selecting a larger proportion of high document frequency (i.e. common) concepts. Configuration *B* (upper right) illustrates that when common words are discarded by a stricter 2.5% document frequency limit, the difference in concept recall reduces substantially. In both configurations *A* and *B*, *COMBO* performs as expected, achieving a level of concept recall in between *CTS* and $DM_{nfx}$.

For a larger (500 document) result set sample size all three experimental algorithms achieve slightly less concept recall. A larger sample size appears to slightly reduce the difference in concept recall between *CTS* and $DM_{nfx}$. Under configuration *D* (lower right) the difference is never more than 7%. Given that the average number of concepts per query is about 16, this difference represents less than one concept per query.

Another interesting observation is that *CTS* appears to be less sensitive to document frequency filtering than $DM_{nfx}$. This is a desirable property because determining a specific document frequency filter that works well in all cases may be difficult.

In summary, this section has shown that $DM_{nfx}$ may achieve higher concept recall than *CTS*, but this difference is mostly due to high document frequency terms which we expect to be less useful for query refinement. Moreover, *CTS*'s concept recall appears to degrade slightly less than that of $DM_{nfx}$ for larger result set sample sizes. The experiments also suggest that *CTS* is less sensitive to the choice of document frequency filtering parameter.

### 3.4.5 Precision Improvement

This section repeats the experiment presented in Section 2.4.3 which measured precision improvement of single term query modifications. Each query refinement algorithm is required to generate 50 term suggestions for each of 150 test seed queries. The precision obtained by adding each suggestion to a seed is measured and compared with the precision of the seed query. The experiment will report the average proportion of term suggestions that yield some improvement in precision and the average improvement in precision achieved by the terms that yield some improvement in precision. Query refinement algorithms are not penalized for generating terms that do not improve precision since, as discussed in Section 2.4.4, such terms may in fact be useful for query refinement purposes.

The argument used in Section 3.4.4 to explain why *CTS* may achieve less concept recall than $DM_{nfx}$ can also be applied to explain why *CTS* may achieve less precision improvement. This section demonstrates that despite this reduction in precision improvement *CTS* is still capable of selecting terms that increase precision.

The graphs in Figure 3-15 plot the average percentage of terms that achieve some improvement in precision under each of the four experimental configurations. As before, the horizontal axis represents initial query precision. Each experimental algorithm was asked to select 50 terms for each of the 150 test queries.

In all four experimental configurations the proportion of precision improving terms is substantially less for *CTS* than for $DM_{nfx}$. However, even at the precision intervals where the algorithms differ the most, *CTS* selects many terms that improve precision. Under configuration A at 30-40% precision about 10 (20%) of the 50 terms selected by *CTS* improve precision. Similar results were obtained for the other three configurations. Although in most cases, *CTS* yields a reduction in precision improvement, we consider that it ability to virtually eliminate information loss may warrant the extra effort.

As expected *COMBO* reduces the difference in precision improvement from $DM_{nfx}$. Under configurations *C* and *D* the average percentage of precision improving terms achieved by *COMBO* for queries with initial precisions between 0 and 40 percent fluc-

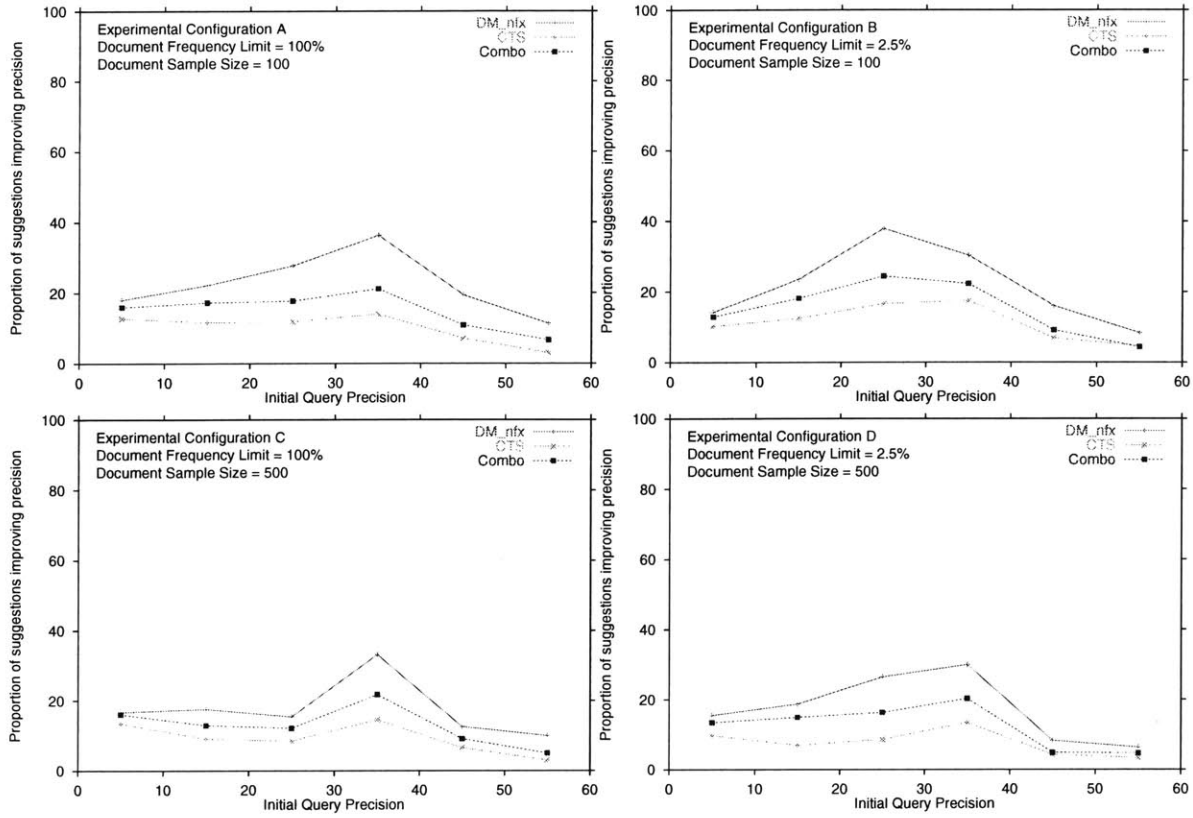



Figure 3-15: Average proportion of terms improving precision versus initial query precision. 84K documents, 150 queries.

tuates between 14% and 20%. This means that out of the 50 terms selected by the algorithm between 7 and 10 terms yield some improvement in precision. It is important to remember that, as explained in Section 2.4.3, terms that do not improve precision may still be useful for query refinement purposes.

The graphs in Figure 3-16 illustrate the average improvement in precision achieved by the terms that yield some improvement in precision. Improvement in precision indicates the difference between the precision of the seed query and the precision of the seed query conjoined with each single selected term. For instance, a 10% improvement in precision for a result sample size of 100 documents means that the augmented query retrieved 10 (10% of 100) more relevant documents than the original seed query.

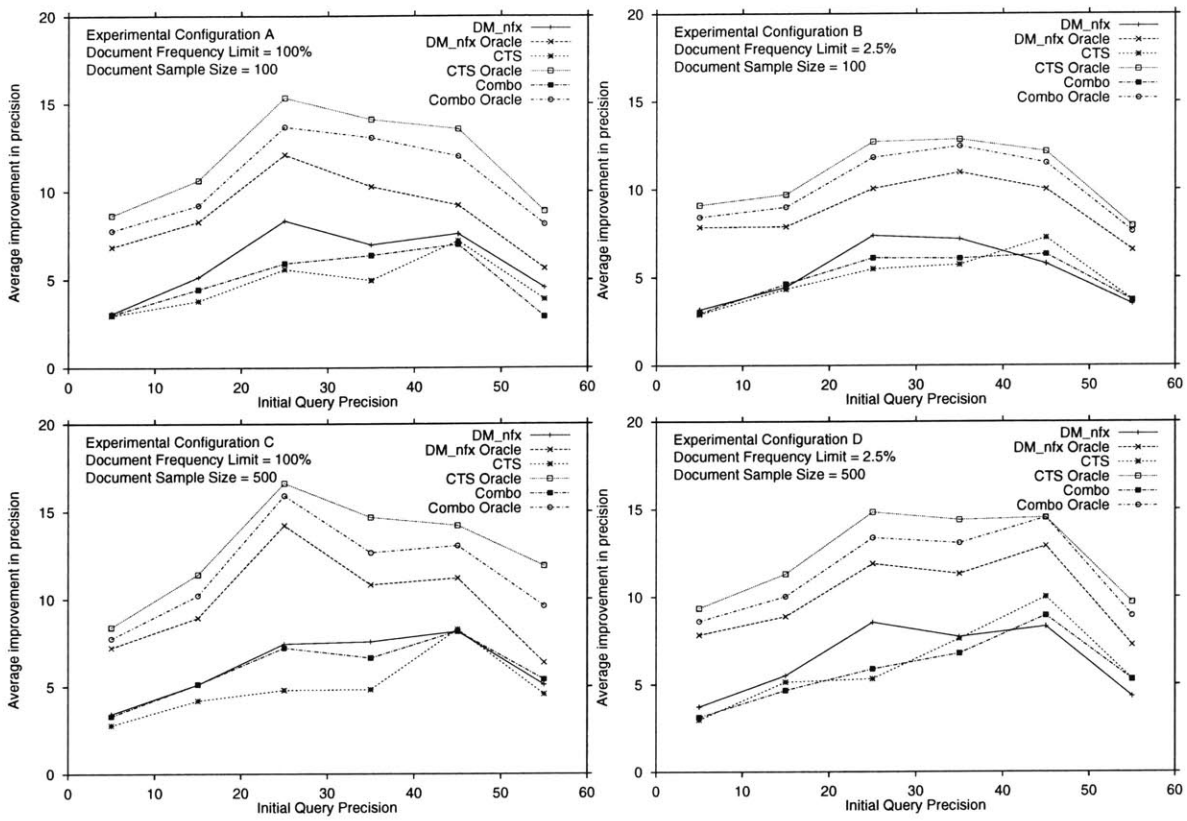



Figure 3-16: Average precision improvement achieved by *CTS*, $DM_{nfx}$, and *COMBO* (84K documents, 150 queries)

The graph for each algorithm consists of two lines, one plotting the precision improvement achieved by the selected terms that improve precision and the other one plotting the maximum average precision improvement achieved by an Oracle algorithm with the same number of terms. This Oracle algorithm was computed by considering all terms appearing in documents in the result set and measuring their improvement in precision (see Section 2.4.3). Although the oracle used for each algorithm is the same, each algorithm has it own oracle line because, although each algorithm selects the same number of terms, the number of precision improving terms may differ. Averages across samples of different sizes would not be directly comparable. The improvement in precision achieved by each algorithm should therefore be interpreted relative to the maximum achievable

by its corresponding oracle. For each initial precision interval the metric that should be used to compare the algorithms should be the difference between the average precision improvement achieved by the algorithm and that achieved by its corresponding oracle.

Figure 3-16 once more confirms that *CTS*, although not as effective as $DM_{nfx}$ in terms of precision improvement, is capable of selecting terms that improve precision by more than 5% in most initial precision intervals. Again, *COMBO* selects terms that yield precision improvement between *CTS* and $DM_{nfx}$. Notice how the lines for *CTS* lie between those of *COMBO* which in turn lie within the lines for $DM_{nfx}$.

### 3.4.6 Performance of *CTS*

In Chapter 3 we demonstrated that ideal coverage term selection is an NP-complete problem. As a result we designed *CTS*, a classic greedy approximation algorithm. In this section, we report on the running time achieved by each of our experimental algorithms.

A total of five algorithms will be tested under the same four experimental configurations used above. In addition to *CTS*, $DM_{nfx}$ and *COMBO*, all of which use our *LOOK* query lookahead algorithm, we will report on two variations of $DM_{nfx}$. The first variation, $DM_{nfx}$(NO QLA), selects terms using the exact same weight function used by $DM_{nfx}$ but performs no query lookahead. The performance of $DM_{nfx}$(NO QLA) relative to $DM_{nfx}$ reflects the overhead of query lookahead using the *LOOK* algorithm over a term frequency-based term selection that does not perform query lookahead. $DM_{nfx}$(Slow QLA) performs lookahead of a refined query by looking up the inverted file entry for the additional term and computing its intersection with the seed query's result set. The performance of $DM_{nfx}$(Slow QLA) relative to $DM_{nfx}$ indicates the amount of time saved by using *LOOK*.

Figure 3-17 shows the average runtime in seconds required by each algorithm to generate 50 terms for each of our 150 test queries. It is important to keep in mind that although only 50 terms are returned, each algorithm may potentially have to lookahead a much larger number of queries. In our tests, the average number of terms extracted

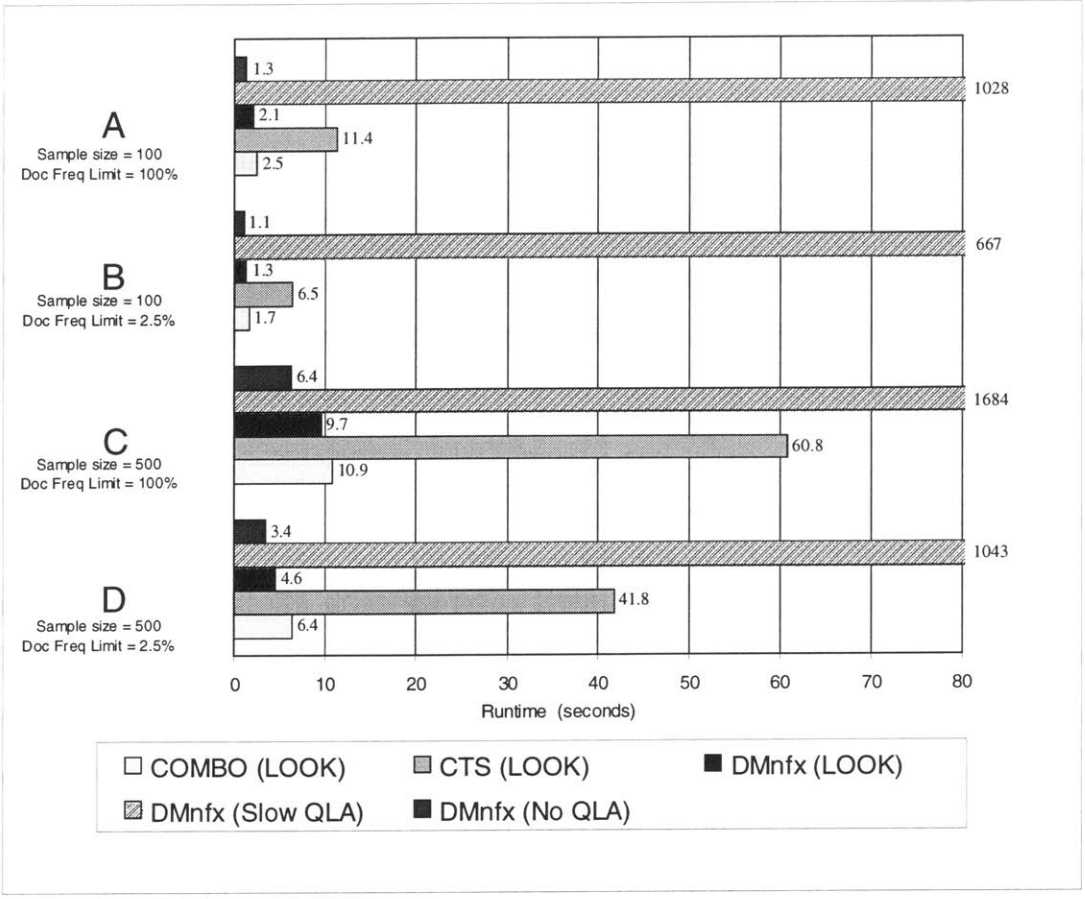



Figure 3-17: Runtime Graph.

from samples of 500 documents was about 7,000 terms. The times were measured by running each algorithm five times, discarding the maximum and minimum times and averaging the remaining three times. The tests were conducted on a virtually unloaded 400MHz dual Pentium PC running the Linux operating system (RedHat 5.2). The storage subsystem consisted of a 45 Gigabyte RAID level 5 disk directly connected to the PC via an ultra-wide SCSI interface.

From Figure 3-17 we can draw the following conclusions:

- *LOOK* makes query lookahead practical. The amount of time required to look up and combine inverted file entries is prohibitive as illustrated by the results for

$DM_{nfx}$(Slow QLA).

- *LOOK* demonstrates that query lookahead can be achieved at a small cost (18% to 52%) relative to plain query refinement. Runtime is not a reason for not incorporating query lookahead into query refinement algorithms.

- *CTS* is still too slow to be used in practice for large result set sample sizes (60.8 seconds for 500 documents), but it is practical for smaller sample sizes. However, a linear implementation of *CTS* appears possible and may fix this problem.

- *COMBO* is a feasible alternative to *CTS* that achieves running times that are 13% (for configuration D) to 30% (for configuration C) more than $DM_{nfx}$. The experiments of Sections 3.4.3 to 3.4.5 demonstrate that *COMBO* can achieve lower redundancy and information loss when compared to $DM_{nfx}$ and better concept recall and precision improvements when compared to *CTS*. Moreover, *COMBO* can provide a control knob that can be used to trade off speed for effectiveness.

## 3.5 Summary

In this chapter we introduced the technique of query lookahead and showed how it could be used to develop coverage term selection for query refinement. Coverage term selection chooses terms that minimize redundancy and information loss by maximizing the proportion of result set documents covered by the chosen terms.

We designed and implemented a fast query lookahead algorithm named *LOOK* capable of computing the result sets of thousands of queries in less than a second on standard PC hardware. *LOOK* demonstrated that query lookahead can be computed at a cost that is within a small factor of the cost of query refinement alone for a fixed result set sample size.

We defined a model of ideal coverage term selection using elements of probability theory and used it to prove that ideal coverage term selection is NP-complete. We then

presented a greedy approximation algorithm that we call *CTS* and demonstrated that it is capable of reducing both redundancy and information loss when compared to previously known frequency-based algorithms as exemplified by the $DM_{nfx}$ algorithm introduced in Chapter 2.

While our experiments demonstrated the superiority of *CTS* in reducing redundancy and information loss, they also showed that *CTS* is too slow to be used in practice for large result set sample sizes. As an alternative we proposed *COMBO*, an algorithm combining the speed of $DM_{nfx}$ with the effectiveness of *CTS* in reducing redundancy and information loss. *COMBO* runs in time comparable to that of $DM_{nfx}$ and achieves significantly less redundancy and information loss than the frequency-based algorithm with a marginal degradation in other effectiveness measures.

In Chapter 4 we will apply query lookahead to develop a new type of user interface that we call *interactive query hierarchies*. Interactive query hierarchies present a hierarchically organized view of a result set and are unique in their use of queries as categories upon which result set documents are organized.

# Chapter 4

# Interactive Query Hierarchies for Result Set Visualization

## 4.1 Introduction

In this chapter we introduce *interactive query hierarchies*, a new user interaction paradigm integrating searching and browsing. An *interactive query hierarchy* is a tree of refined queries together with their individual result sets. A user provides an initial *seed* query. Refined queries combine terms appearing in the result set with the terms in the seed query. The descendants of a query provide a succinct hierarchical categorization of the collection of documents matching that query.

In Chapter 3 we showed how *LOOK*, an efficient query lookahead algorithm, could be used to implement *CTS*, a coverage term selection algorithm. To achieve performance, *LOOK* exploits the fact that *CTS* can be implemented by eagerly evaluating refined queries that conjoin term suggestions with a seed query. This chapter will demonstrate how *LOOK* can be used to eagerly compute the result sets of refined queries in an interactive query hierarchy.

The diagram in Figure 4-1 shows a two-level interactive query hierarchy with `space` as the seed query and three child queries: (`space` $\wedge$ `mir`), (`space` $\wedge$ `nasa`) and (`space`

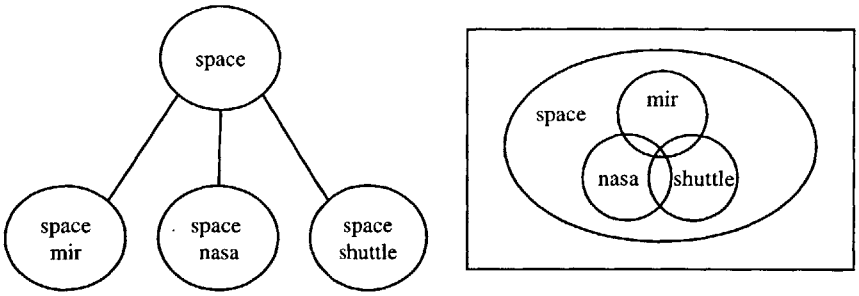



Figure 4-1: A two-level query hierarchy for the query space together with a Venn diagram depicting the result set of each query.

$\wedge$ shuttle). In this example, each child query combines a single additional term conjunctively with the seed query. Also shown in the figure is a Venn diagram displaying the results set of each query. The result sets of the child queries organize the result set of their parent in three clusters of documents. Each child result set contains the subset of documents matching its corresponding query.

Interactive query hierarchies present an interesting alternative to the type of user interface exemplified by the HyPursuit system and discussed in Chapter 2. HyPursuit offers the user more control over the formulation of refined queries at the cost of reducing the amount of information computed in advance. Interactive query hierarchies limit the types of refined queries that are suggested to the user but increases the amount of information about the consequences of selecting among such queries.

Document clustering approaches like Scatter/Gather [61] also provide automatic result set visualization but lack a precise and predictable semantics for what constitutes a cluster. Cluster descriptions do not always convey a correct description of their contents. These clustering tools typically compute a matrix of similarities between each pair of documents. Using this matrix, document that are similar to each other are grouped into clusters. Once clusters are formed, a compact description of each cluster, also known as a cluster digest, is generated by extracting terms and other information from the documents in the cluster.

Although document clustering systems can prove to be quite useful in many circumstances they suffer from a number of shortcomings limiting the scope of their usefulness:

- good compact descriptions of clusters are difficult to generate
- documents are typically associated with only one cluster, although several cluster descriptions may be relevant to the same document
- documents do not necessarily end up in the cluster whose description seems most relevant
- it is difficult to compare the quality of clusters generated by different algorithms

Key to our design of interactive query hierarchies is the use of queries to specify the categories upon which documents of a result set can be classified. One advantage of this approach is that categories inherit their semantics from queries. For instance, the categories that a given document belongs to are unambiguously determined by the semantics of the query language used. Documents belong to all the categories (queries) that match the document. In contrast to agglomerative clustering, interactive query hierarchies do not force the result sets of child queries to partition the result set of their parent node. A given document may appear in the result set of more than one child query. In this last respect, our approach is more akin of the "soft clusters" introduced by Pereira et al. [45]. Their soft clustering approach, however, still inherits the semantic ambiguities of agglomerative clustering.

As we will show in this chapter, queries themselves can serve as extremely compact category descriptors. This can be achieved by organizing queries in a hierarchy. Each level in the hierarchy refines the upper layers by adding one or more terms to the parent query. The query representing each category can be obtained by traversing the path from the seed query to the desired category. Each query can be labeled with the part of its query that distinguishes it from its ancestors.

In this chapter we present a graphical user interface (GUI) supporting interactive query hierarchies. Our GUI incorporates the notion of *search sessions* during which users can submit several refined queries and examine their results without losing track of their intermediate results. An interactive query hierarchy can be expanded and contracted allowing a user to explore alternate new search paths without losing track of any potentially relevant information already found. Many existing query refinement user interfaces force a user to resubmit refined queries and make it difficult to return to a previous state in the refinement process.

In summary, interactive query hierarchies improve previous query refinement user interfaces in three ways:

- they provide a hierarchical classification of a seed result set that uses queries as classification categories
- they provide instant access to the result sets generated by refined queries without requiring iterative re-submission of queries to a search engine
- they allow a user to navigate the hierarchical classification in all directions while maintaining the entire context of the search session

An interactive query hierarchy offers a hierarchy of automatically refined queries that enables the user to quickly pursue more focused searches. These refined queries narrow the focus of the user query and can help the user quickly formulate a precise specification of their information need. Interactive query hierarchies provide progressively finer descriptions of the pertinent information subspaces. Our new interaction model advances the state of the art in information retrieval technology by automatically providing, from a simple user query, a compact and browsable organization of a potentially large and imprecise result set.

Using query lookahead, interactive query hierarchies allow a user to immediately examine the result set of a refined query without having to repetitively submit refined queries to the search engine. This improves response time as well as search engine

utilization, since it minimizes the amount of computation spent unnecessarily evaluating queries from scratch. As will be shown in Section 4.5, our implementation of query hierarchies only requires evaluation of a query from scratch once for the seed query.

This chapter reports on our experience implementing *Inforadar*, a complete network search system supporting a user interface with interactive query hierarchies. The *Inforadar* system consists of a search engine server, an indexing module and a client applet. *Inforadar* supports a Boolean query language extended with document ranking inspired on the extended Boolean query model [50].

The *Inforadar* applet can process multiple user interface operations concurrently. This means that, for instance, while the user waits for an hierarchy expansion request to be processed he/she may continue to browse the query hierarchy or perform other user interface operations. Allowing concurrent user interface operations has the potential for hiding the latency involved in processing individual requests. Hiding this latency is of particular importance to us, because our search engine must spend a substantially larger amount of effort than the typical search engine in order to compute the query hierarchies.

In addition to providing a complete description of the system, this chapter describes the most important design decisions behind each major component of the system. The remaining of this chapter is organized as follows:

Section 4.2 describes the query model supported by *Inforadar*.

Section 4.3 describes the *Inforadar* graphical user interface.

Section 4.4 discusses the query hierarchy generation algorithm.

Section 4.5 discusses the design and implementation of *Inforadar*.

Section 4.6 presents performance measurements and then reports on a small pilot user study comparing *Inforadar* with a more traditional user interface.

Section 4.7 summarizes the major findings of this chapter.

## 4.2 Query Model

So far in this dissertation we have purposely avoided tying our applications of query lookahead, coverage term selection and interactive query hierarchies, to any particular query model. Doing so would have unnecessarily restricted the generality of the ideas. In this section we describe the query model adopted by *Inforadar* as part of the complete description of our prototype search system. Our choice of query model does not preclude other models from taking advantage of the contributions of this dissertation. We also present the rationale behind our choice of query model.

Several query models have been proposed in the past. Many of these models are hybrids of two fundamentally different models: the Boolean query model and the vector query model. Boolean queries (see Definition 3.1) are predicates combining terms using logical operators $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT). A document matches the query if it satisfies the logical predicate. Vector queries are flat lists of weighted terms. A document matches a vector query if it contains at least one of the terms in the query.

Query refinement algorithms require a means of ranking documents according to their belief relevance. Ranking is useful to select a sample of documents from the seed result set to extract term suggestions (see Section 3.3). The ranking mechanism is dependent on the query model chosen. For instance, pure Boolean queries provide no means for ranking documents. A document either matches the query or it does not. In the vector model the matching documents are typically ranked using some sort of query-to-document similarity function measuring how close the query resembles each matching document. Resemblance is usually proportional to the number of terms that the document and query have in common. The reader is referred to Zobel [70] and Grossman [29] for recent comprehensive treatments of alternative document ranking techniques for vector queries.

A compromise between strictly Boolean and vector queries is also possible. An example of such a query model is the *extended Boolean model* [50, 53]. This model provides a mathematically sound mechanism for ranking the result sets of general Boolean queries. The details of the original model are quite complex, but the main idea is simple. The

model defines a similarity function that takes into account the logical operators used in the query. For conjunctive (AND) queries, the function gives precedence to documents containing a larger proportion of the terms in the query. For disjunctive queries, the similarity function gives preference to documents containing a potentially smaller fraction of the query terms but with higher weights.

The structure of an extended Boolean query resembles that of a Boolean query. The difference is that in an extended Boolean query, each subquery can be assigned a weight. As in the vector model, weights encode the relative importance of one part of the query versus another. Section 1 gives examples of boolean, vector and extended boolean queries.

In addition to determining the document ranking scheme, the query model plays a fundamental role in the generation of the refined queries used to construct interactive query hierarchies. The model dictates the possible ways that new terms can be combined with a seed query to form refined queries. Boolean refined queries can be generated by forming a valid logical combination of new terms with the terms in the seed query. Vector refined queries can be formed by simply adding new terms to the seed query or replacing old terms with new ones.

*Inforadar* adopts a simplified extended Boolean query model that combines Boolean operators with document ranking. The simplifications are both semantic and syntactic. The first semantic simplification is that *Inforadar* disallows nested Boolean queries. The second semantic simplification is that our model does not allow weights to be specified for the different parts of a query. The model assumes that all subqueries of a query have the same weight.

Syntactically, *Inforadar* adopts the format used by various popular Internet search engines (e.g. www.altavista.com, www.hotbot.com, www.lycos.com). The syntax uses plus (+) and minus (-) signs to specify logical operations. A query is just a flat list of terms where each term could be preceded by a plus or minus sign. In order to match a query, a document must contain all the terms with plus signs, but must not contain any terms with minus signs. Terms without plus or minus signs may or may not appear in

a matching document. In our model, the query `(+space +shuttle -mars)` matches the same set of documents as the Boolean query `(space ∧ shuttle ∧ (¬ mars))`.

The semantics of *Inforadar* queries can be formally specified by a de-sugaring operator that transforms *Inforadar* queries into Boolean queries. In the style of [64], the following de-sugaring operator $\mathcal{D}$ takes a sequence of plus ($p_i$'s), minus ($m_i$'s) and plain ($o_i$'s) terms and transforms it into a semantically equivalent Boolean query. The Boolean values true and false are used to maintain correctness in the presence of *Inforadar* queries consisting of a single plus or plain term. *Inforadar* does not allow queries consisting of a single term preceded with minus sign.

$$\mathcal{D}[\![(+p_1 \cdots + p_k \quad - m_1 \cdots - m_l \quad o_1 \cdots o_m)]\!] = \\ (\text{true} \wedge p_1 \wedge \cdots \wedge p_k \wedge \neg(m_1 \vee \cdots \vee m_l) \wedge (\text{false} \vee o_1 \vee \cdots \vee o_m)) \quad (4.1)$$

Document ranking in *Inforadar* is simplified by the absence of nested queries. Once the result set of a query has been computed, document ranking does not have to consider terms preceded by minus signs. By definition, these terms do not appear in any of the documents in the result set. The remaining plus and plain terms can be treated as a vector query for the purposes of ranking documents. As explained before in this section, many different document ranking approaches for vector models have been proposed in the past. Document ranking functions are some of the most carefully guarded secrets of commercial Internet search engines.

After several experiments with various document ranking approaches, we selected the one that achieved the best overall performance (highest precision and recall) in our test collection. Surprisingly, the winning approach turned out to be the simplest. *Inforadar* assigns to each document a weight equal to the number of query terms appearing in the document. In case two documents have the same number of query terms, the document with more occurrences of those query terms is chosen. The rank of a document is its position in the result set sorted by decreasing weight. More sophisticated approaches yielded

worse or not significantly better performance. For the extremely short test queries (average of 2 terms) that we use, it appears that other factors can easily overwhelm the importance of the presence of queries terms. For instance, when we added document size normalization to the query-to-document similarity function, this resulted in a significant reduction in both precision and recall. In our test collection, although the average document is relatively short (200 unique terms), a small (percentage-wise) difference in document size between two documents may easily overcome a large (percentage-wise) difference in the number of query terms present in these documents.

Although there are many choices for constructing refined queries under our query model, we wanted to take advantage of the results of Chapter 3 and incorporate the query refinement algorithm developed in that chapter. *Inforadar* uses fast query lookahead (*LOOK*) to eagerly compute the result sets of refined queries. To gain efficiency, *LOOK* assumes that the refined queries conjunctively combine refinement terms with the seed query. This allows *LOOK* to evaluate each refined query without examining documents outside the result set of the seed query. Although conjunctive queries may not be optimal in many circumstances, the efficiency gained is substantial and may well be the difference between a practical query hierarchy system and an impractically slow one. Also conjunctive queries have the added benefit of reducing the search space as terms are added to the seed query.

There are two additional advantages of adopting the simplified extended Boolean model described in this section. First, the syntax is simple and avoids the complexity of nested Boolean queries. The flatter queries of our model more closely resemble a natural language description of an information need. Second, the chosen model is extremely popular among users of commercial Internet search engines. This will facilitate the transition of our ideas into production systems. A popular query model will also allow us to conduct user studies, since subjects will require less training.

We do not want to end this section without emphasizing again that the main ideas presented in this dissertation, including query lookahead, coverage term selection and

interactive query hierarchies, do not preclude other query languages. Where appropriate in the remainder of this dissertation we will discuss the applicability of our ideas to other query models.

## 4.3 Graphical User Interface (GUI)

This section describes the graphical user interface (GUI) supported by *Inforadar* with special emphasis on its support for interactive query hierarchies. The interactive functionality of the *Inforadar* GUI is implemented by an applet written in the Java programming language [28]. The applet communicates with the *Inforadar* search engine server through a protocol designed on top of the Common Gateway Interface (CGI) [10]. A more detailed description of the *Inforadar* system design, including the search engine, the communication protocol and the indexing module, will be presented in Section 4.5.

The examples presented in this section use a document database consisting of about eighty thousand Associated Press articles from 1989. This collection of press articles is a subset of the TREC (trec.nist.gov) test document collection [33] and was also used in some of the experiments presented in Chapter 2 and Chapter 3.

The *Inforadar* client applet consists of about three thousand lines of Java code and was developed using the Java Software Development Kit version 1.2. The development of the applet was greatly simplified by the use of several GUI software components provided by Java Foundation Classes (JFC) [17]. The GUI components accounting for most of these savings were the *JTree* class, used to implement a hierarchical display of queries, the *JScrollPane* class, used to implement windows with scrolling capabilities, and the *java.net* package, used to implement the client/server communication layer. Thanks to the JFC, a team of three freshman students with no previous experience with Java was able to complete the first fully functional prototype applet in about six weeks.

The *Inforadar* applet is triggered when a web browser retrieves a web page containing an appropriate applet directive. Initially, the applet displays the window in Figure 4-2.

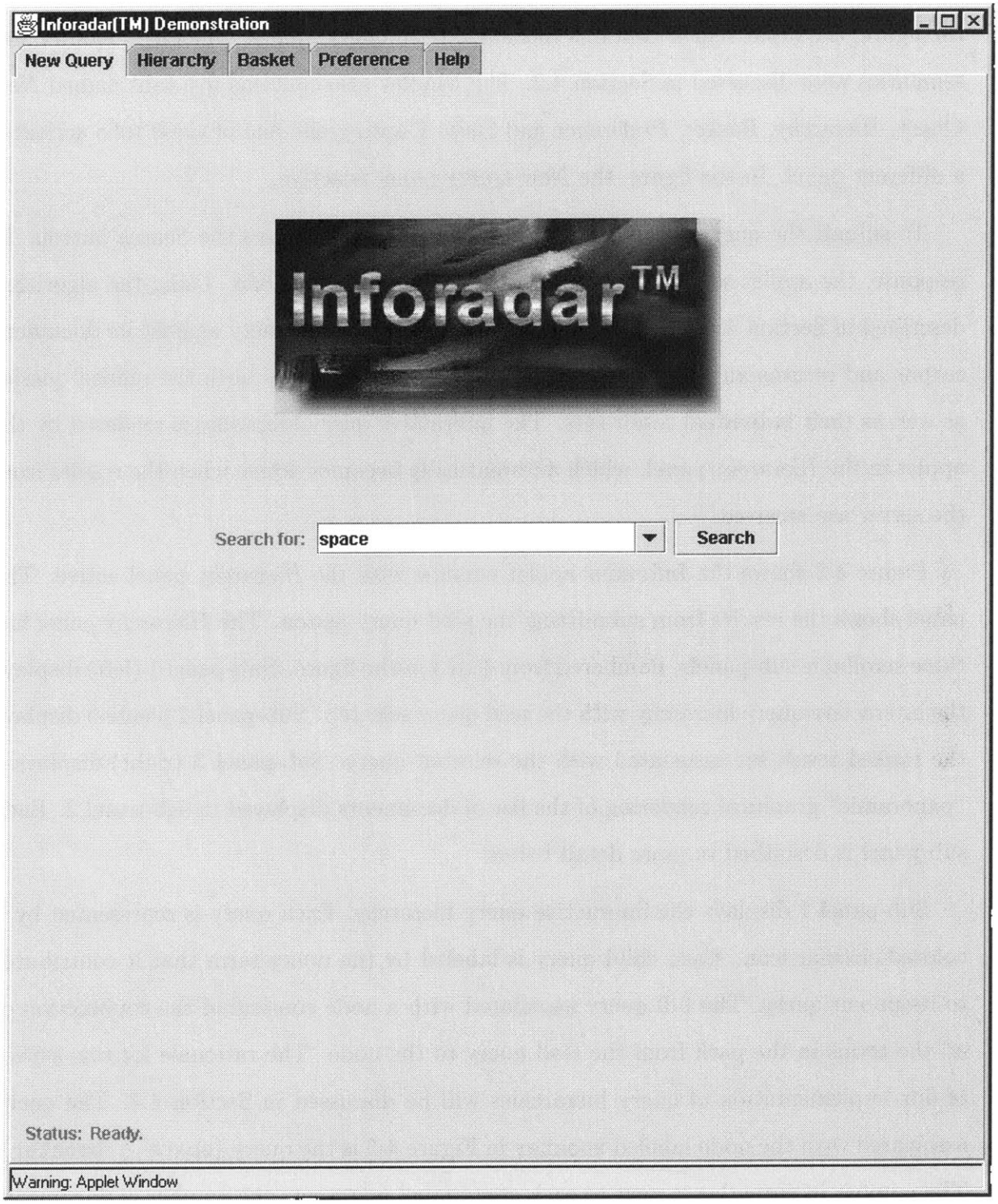



Figure 4-2: The *New Query* panel.

This window has the customary text input region where a user can enter a query. In the figure, the input region contains the single-term query `space`. The query syntax and semantics were discussed in Section 4.2. The window also contains five tabs named *New Query*, *Hierarchy*, *Basket*, *Preference* and *Help*. Clicking one one of these tabs activates a different panel. In the figure, the *New Query* panel is active.

To submit the query entered in the text region a user presses the *Search* button. In response, the applet sends the query to the search engine module. Using the algorithm described in Section 4.4, the search engine evaluates the seed query against its document corpus and returns an interactive query hierarchy that includes both the refined queries as well as their individual result sets. The interactive query hierarchy is rendered by the applet in the *Hierarchy* panel, which automatically becomes active when the results from the server are received.

Figure 4-3 shows the *Inforadar* applet window with the *Hierarchy* panel active. The panel shows the results from submitting the seed query `space`. The *Hierarchy* panel has three scrollable sub-panels, numbered from 1 to 3 in the figure. Sub-panel 1 (left) displays the interactive query hierarchy with the seed query selected. Sub-panel 2 (center) displays the ranked result set associated with the selected query. Sub-panel 3 (right) displays a "panoramic" graphical rendering of the list of documents displayed in sub-panel 2. Each sub-panel is described in more detail below.

Sub-panel 1 displays the interactive query hierarchy. Each query is represented by a colored circular icon. Each child query is labeled by the query term that it contributes to its parent query. The full query associated with a node consists of the conjunction of all the terms in the path from the seed query to the node. The rationale for this aspect of our implementation of query hierarchies will be discussed in Section 4.4. The query associated with the node labeled `booster` in Figure 4-3 is the query (`space` $\wedge$ `booster`). The number in parentheses next to each query label represents the number of documents in the result set specific to the corresponding query. For instance, the query (`space` $\wedge$ `booster`) has 85 matching documents among the 500 documents in the result set sample

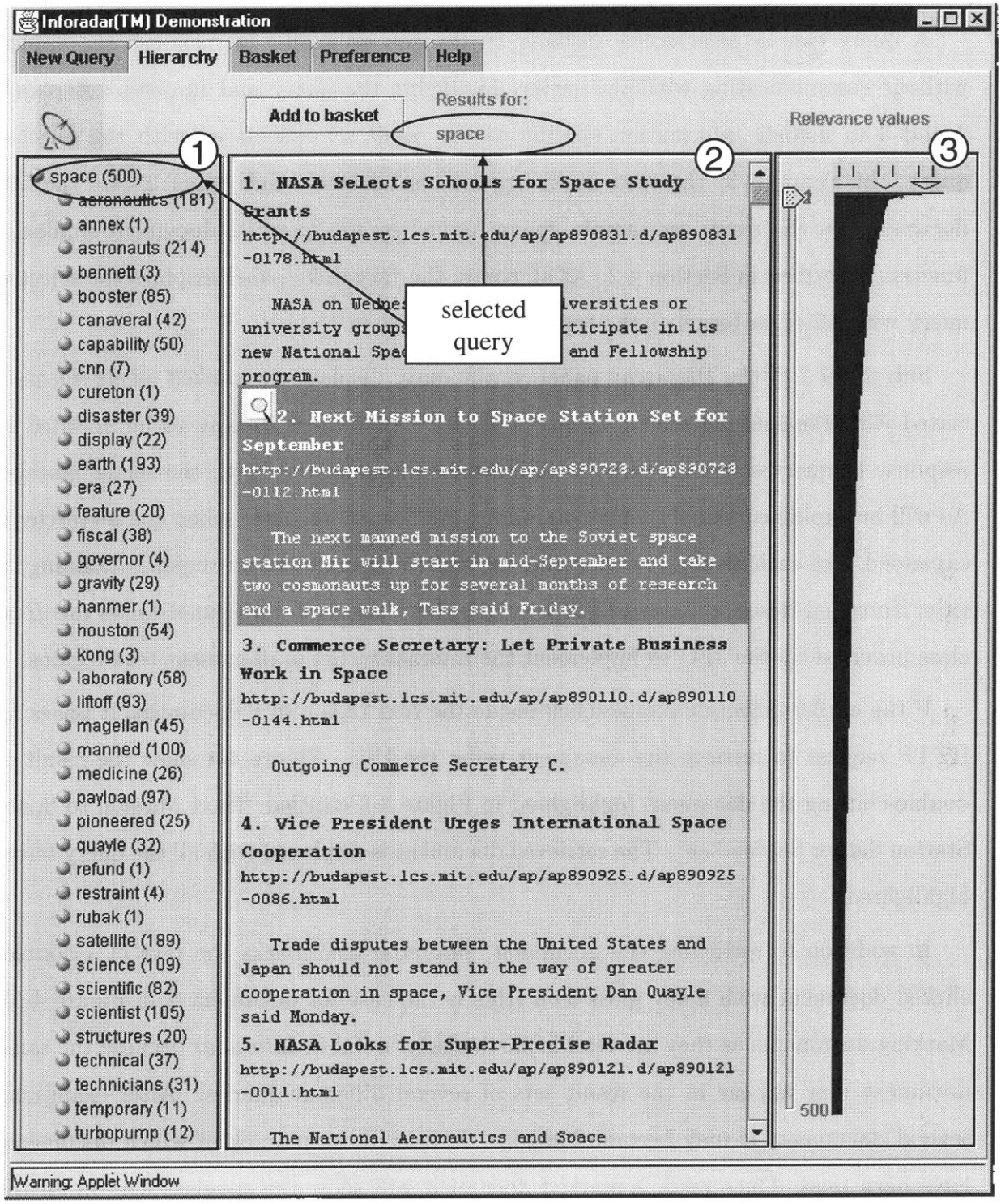



Figure 4-3: The *Hierarchy* panel with an interactive query hierarchy for the query `space`

for the seed query space.

A query can be *selected* by clicking on its icon or label. In response the applet, without communicating with the server, highlights the query and updates sub-panels 2 and 3 to include information specific to the result set associated with the selected query. In Figure 4-3, the seed query (space) is selected. Sub-panel 2 lists the 500 documents in the result set sample space ranked by the query-to-document similarity function described in Section 4.2. At all times, the *Hierarchy* panel displays the selected query with all of its terms at the top of sub-panel 2.

Sub-panel 2 of the *Hierarchy* panel continuously displays the ranked result set associated with the selected query. Query lookahead allows this display to be updated in response to query selection without additional communication with the server module. As will be explained shortly, such communication is only required when the hierarchy is expanded. For each result set document, sub-panel 2 displays a text region containing its title, Universal Resource Locator (URL) and a short abstract. Sub-panel 2 uses the *JList* class provided by the JFC to implement the interactive list of document text regions.

If the applet detects a double-click inside the text region of a document, it issues an HTTP request to retrieve the document using the URL. Figure 4-4 show the result of double-clicking the document highlighted in Figure 4-3 entitled "Next Mission to Space Station Set for September". The retrieved document is displayed with all the query terms highlighted.

In addition to retrieving the document, *Inforadar* also marks the text of a double-clicked document with a spy glass icon (this is the case for document 2 in Figure 4-5). Marking documents as they are read is particularly useful in *Inforadar* because the same document may appear in the result sets of several different queries. After examining several documents it may become hard for the user to keep track of which documents have been read. Once read, a marked document will show the spyglass icon in all the result sets where it appears until the applet exists.

Sub-panel 3 of the *Hierarchy* panel displays a *similarity graph* for the result set cor-

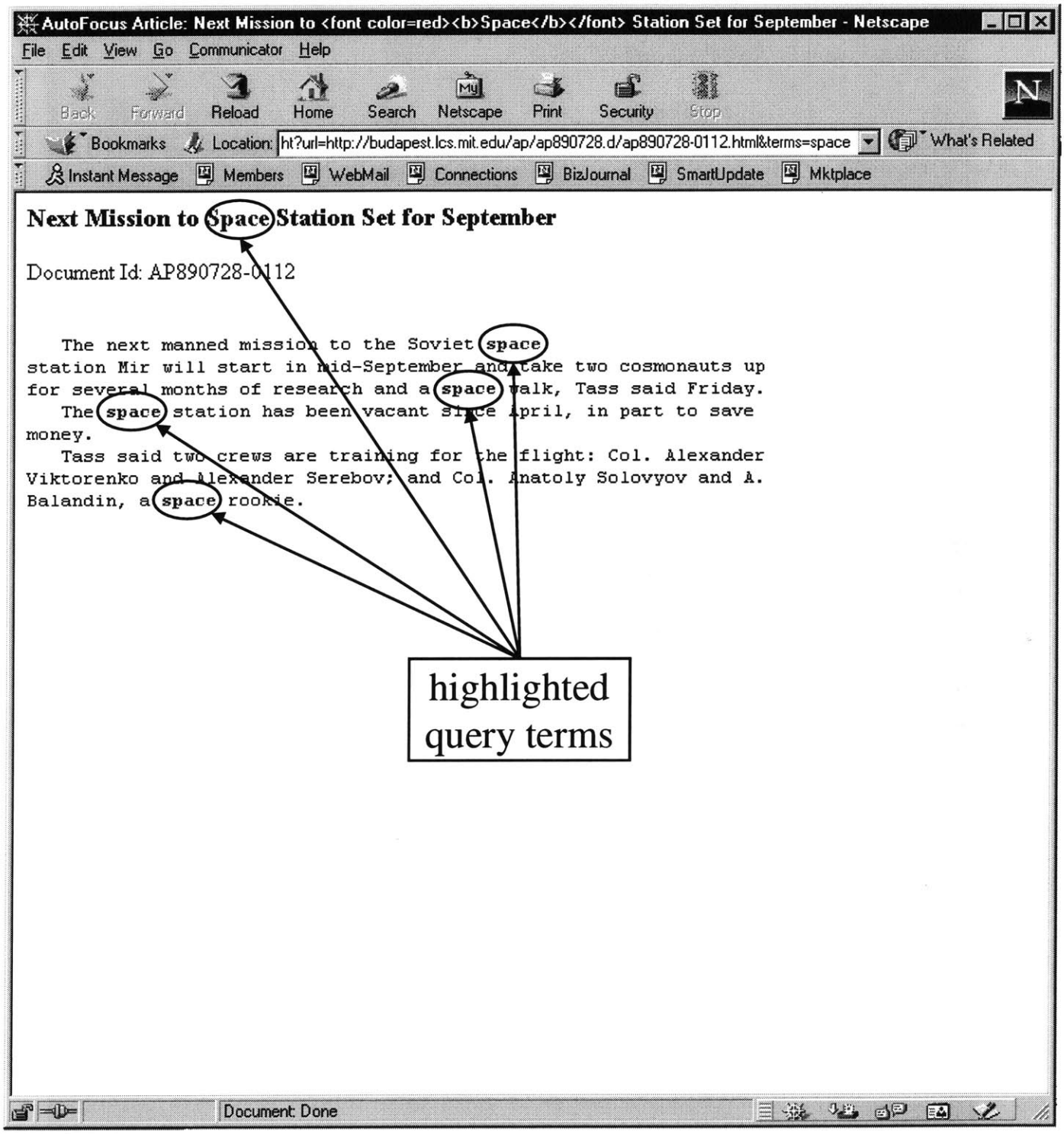



Figure 4-4: Contents of article retrieved by double-clicking document 2 in the window from Figure 4-3. All occurrences of terms appearing in the query are highlighted.

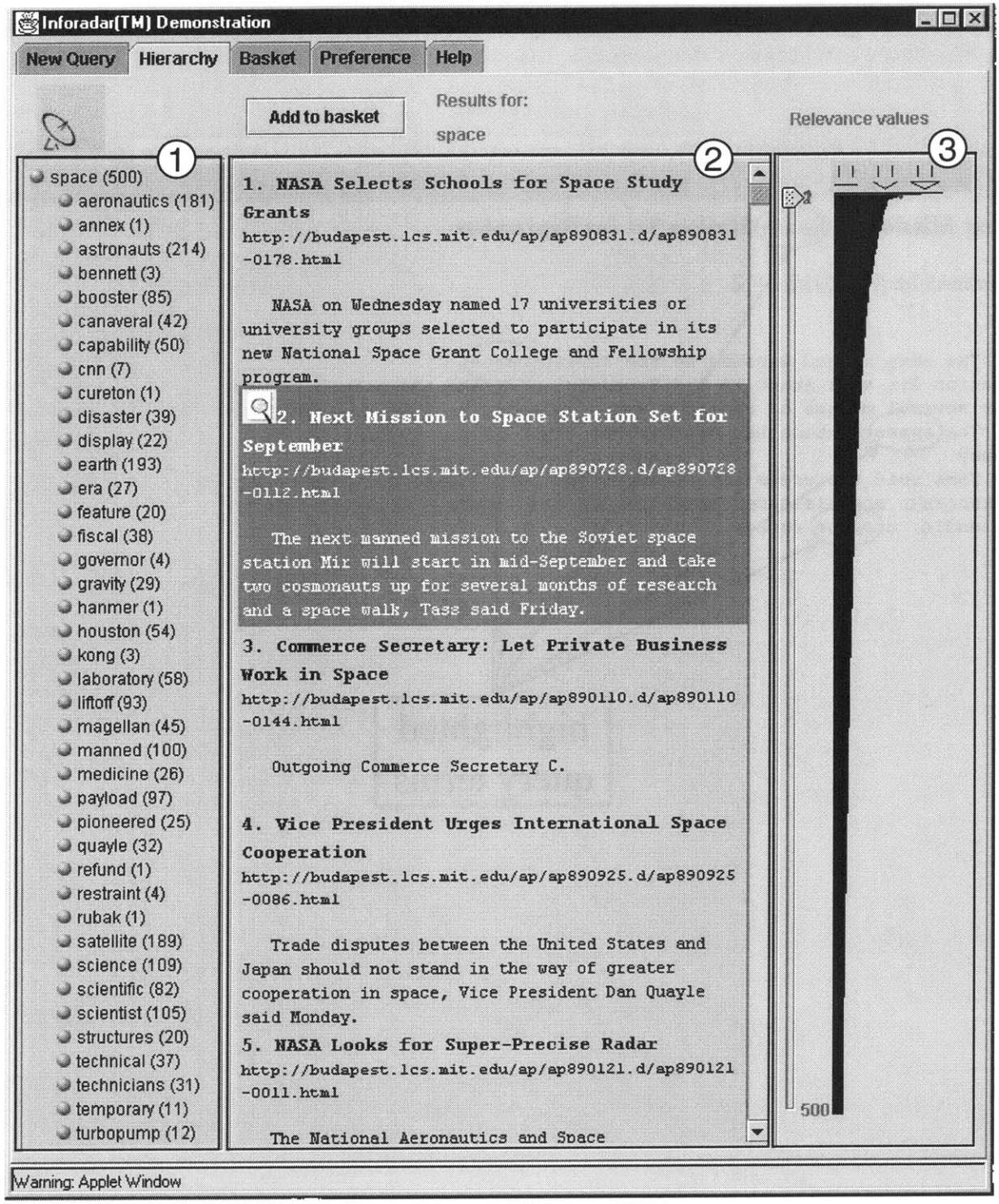



Figure 4-5: The *Hierarchy* panel after examining (double-clicking) document 2 in the window from Figure 4-3

responding to the selected query. This type of graph constitutes our first attempt to provide a graphical "panoramic" view of a potentially large result set. The similarity graph is essentially a bar graph rotated ninety degrees clockwise and containing one bar for each document in the result set. The height of each bar is determined by the similarity between the document and the selected query using the same ranking function used to sort the documents in sub-panel 2. Our similarity graphs were motivated by previous work by other researchers on result set visualization including Veerasamy [66] and Hearst (TileBars) [34].

The user can scroll down the result set for the selected query using either the slider associated with sub-panel 2 or the slider associated with the similarity graph. Either way the applet keeps both the result set (sub-panel 2) and the similarity graph (sub-panel 3) synchronized. For instance, in Figure 4-3 both sub-panels show that document 2 in the result set for the seed query `space` is selected. We are currently investigating how these types of panoramic graphical displays of query result sets can improve information visualization. We hypothesize that these views may uncover important properties that can speed the task of sifting each result set. In particular, we are searching for graphs that could suggest to the user how far down to examine a large ranked result set. Since this is very much work in progress, we will not discuss this aspect of the user interface any further.

Figure 4-3 illustrates how an interactive query hierarchy can organize a large result set (500 documents) into categories made up of refined queries. The set of documents associated with each category is unambiguously determined by the query language; it is equal to the result set for the query making up the category. The interactive query hierarchy shown in Figure 4-3 exposes several interesting subtopics related to the seed query `space`. Some examples of informative terms include `disaster` and `magellan`. Other terms look less informative and less likely to be useful.

There are five operations that can be performed on a query hierarchy node namely: `select`, `expand`, `collapse`, `delete` and `make-root`. As explained before, a node can be

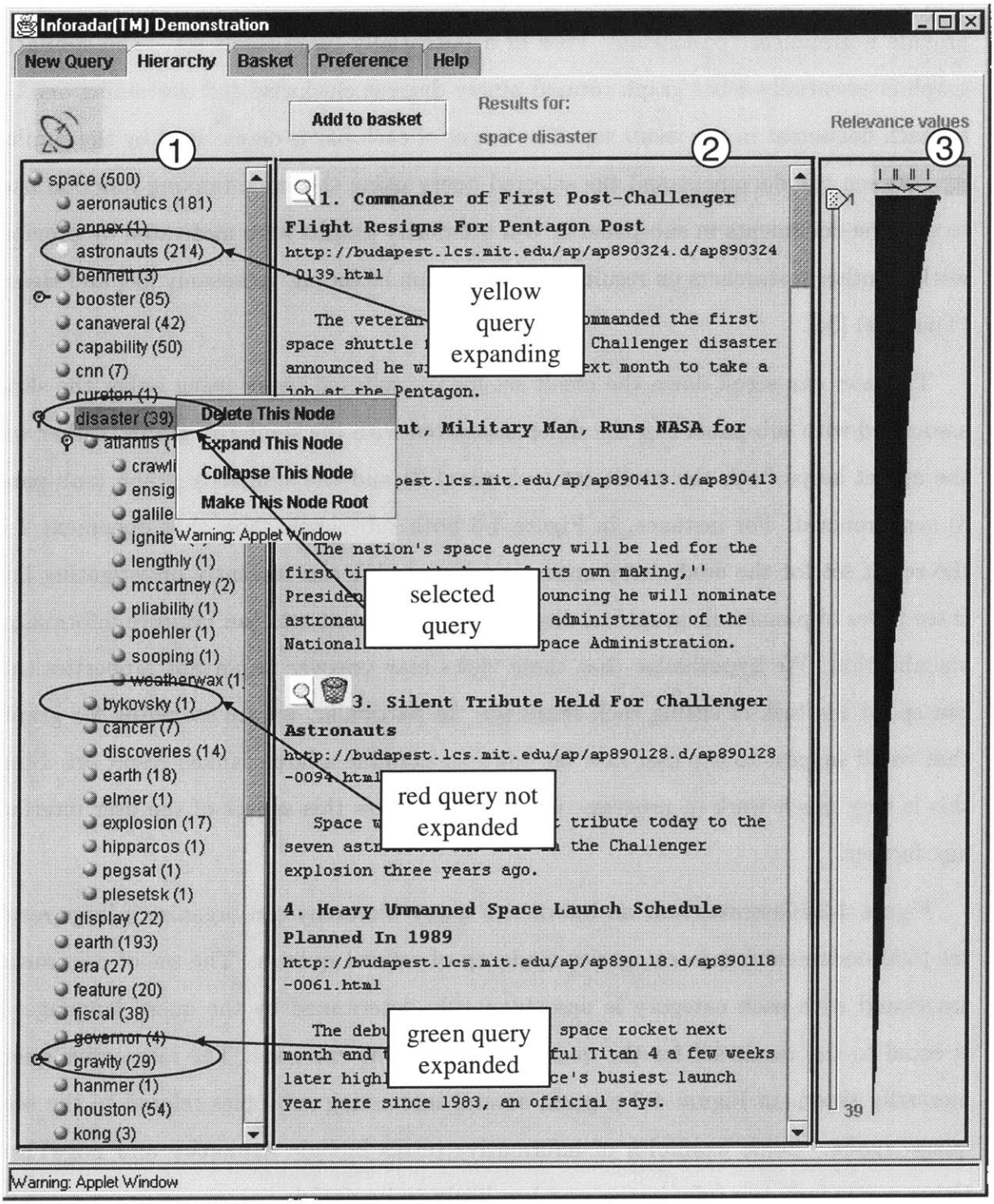



Figure 4-6: The *Hierarchy* panel with the same query hierarchy from Figure 4-3 after several mouse operations.

`selected` with a single left-click of the mouse on its label or icon. All other operations on query nodes can be selected from a menu (see Figure 4-6) that pops up when the user right-clicks on a target node. Figure 4-6 shows the same query hierarchy displayed in Figure 4-3 after several query node operations have been performed.

`Expand`, which can also be triggered by double-clicking on the query label, may have one of two outcomes depending on whether the node has been expanded before or not. *Inforadar* uses colors to indicate the status of a query node. Green nodes have been expanded before. Red nodes have never been expanded. Nodes are temporarily colored yellow while the expansion request is being processed. The first time a node is expanded, the applet sends a request to the server to generate child queries for the selected node. The server treats a query expansion the same way it treats a `Search` button request for the initial seed query. This time, however, the server uses the result set of the parent query, computed by a previous request, as the basis for generating child queries. The applet is responsible for sending the parent result set as part of its request to the server. This guarantees that the semantics of the query hierarchy persist across expansion requests. In particular, this guarantees that the result sets of child queries are always computed relative to the result set of their parents.

Double clicking a previously expanded query alternates between exposing and hiding its child queries. The `collapse` operation simply hides the children of the corresponding node. Special icons are used to indicate whether a node has exposed or hidden children. No icon is shown when the node has never been expanded before. Figure 4-6 shows that the nodes labeled `booster`, `disaster` and `gravity` have been expanded, but only the nodes labeled `disaster` and `atlantis` have visible children.

`Delete` permanently removes the node from the hierarchy. This operation can be used to prune out nodes that are either uninteresting, uninformative or redundant. `Make-root` forces the selected node to become the seed of a fresh interactive query hierarchy. This entails a new request to the server and is equivalent to typing the query associated with the node at the *New Query* panel and pressing the `Search` button. A request to the server

is necessary because the result set for the query becoming root may include documents that were not in the result set sample selected for the previous seed query. The `make-root` operation provides an easy shortcut to users who want to focus on a single sub-query of interest.

The *Inforadar* client applet is persistent which means that it maintains state about all previous results of user interface operations from the duration of the client session. The client session lasts from the time the applet is first loaded by the browser to the time when the applet is terminated.

By maintaining the state of previous user operations during a session, the *Inforadar* applet allows a user to expand and navigate a query hierarchy in any direction without losing track of the information context provided by the preliminary results obtained so far. In Figure 4-6 the results of queries like (`space` $\wedge$ `astronauts`) near the seed query remain accessible even after several queries have been expanded. A user can examine the results associated with different nodes in any order without being forced to remember the results of operations completed before. This inconvenience is common to many search engines supporting query refinement, like Altavista (www.altaivista.com) and Excite (www.excite.com)

Applet persistence is also exploited by the *Inforadar* document basket mechanism. The mechanism allows a user to collect documents of interest in a "basket" by clicking on the `Add to basket` button at the top of the *Hierarchy* panel. Documents that have been added to the basket appear marked with a basket icon in sub-panel 2 (see document 3 in Figure 4-6). The *Basket* panel shown in Figure 4-7 allows a user to examine, modify and print the list of documents collected in the basket. The document basket is particularly useful because it persists across multiple queries. As with the spy glass icon, use of the basket icon is important because the same document may be visited via different query nodes.

*Inforadar* takes advantage of the multi-threading facilities available in the Java programming language by allowing several user interface operations to be issued simulta-

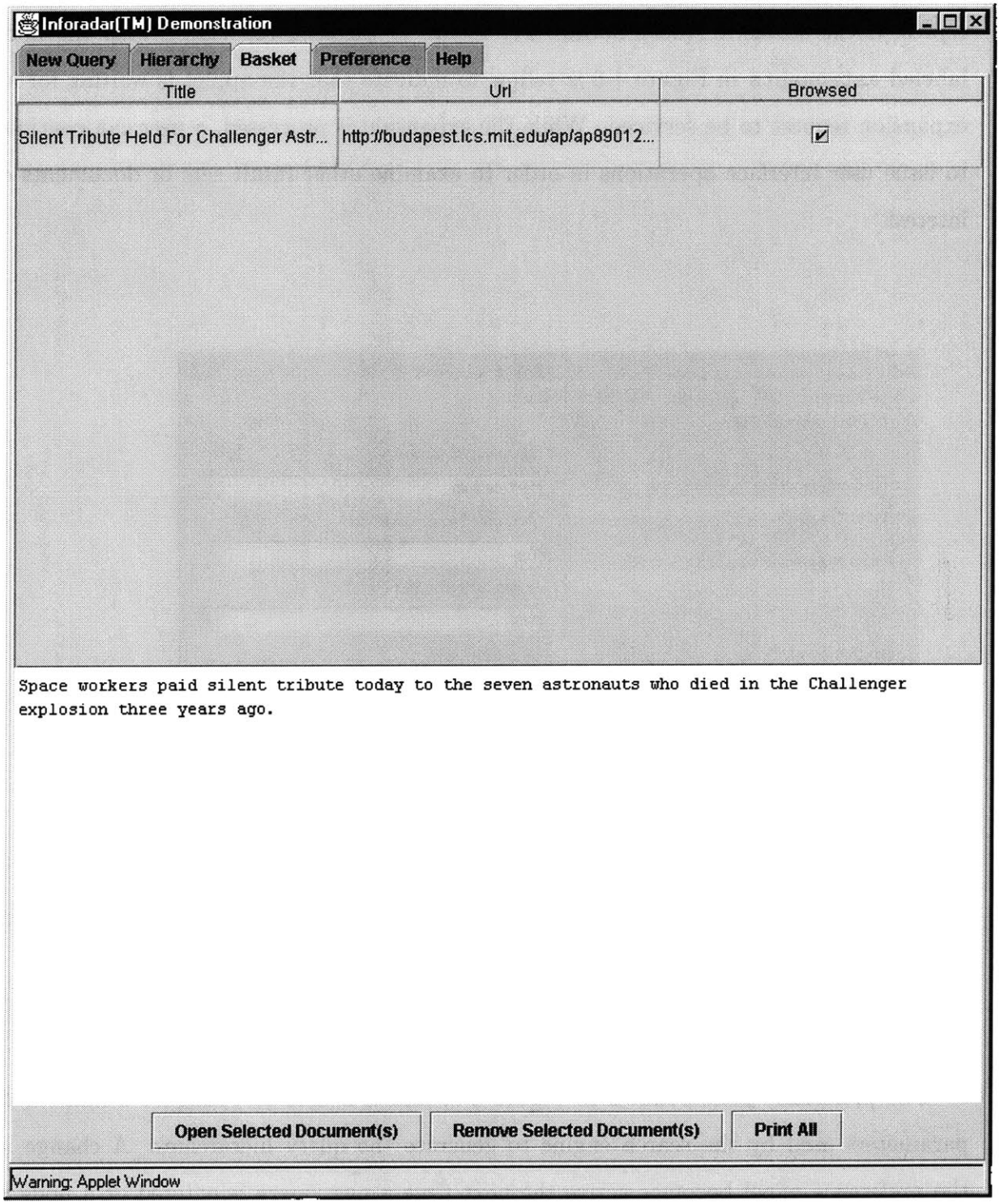



Figure 4-7: The Document Basket panel.

neously. To provide the user with adequate feedback, the applet uses colored icons to represent the status of query nodes. For instance, the icon associated with the node labeled `astronauts` in Figure 4-6 is yellow to indicate that the applet is waiting for an expansion request to be serviced. While the expansion is processed, a user can continue to issue user interface operations in order to examine other result sets or documents of interest.

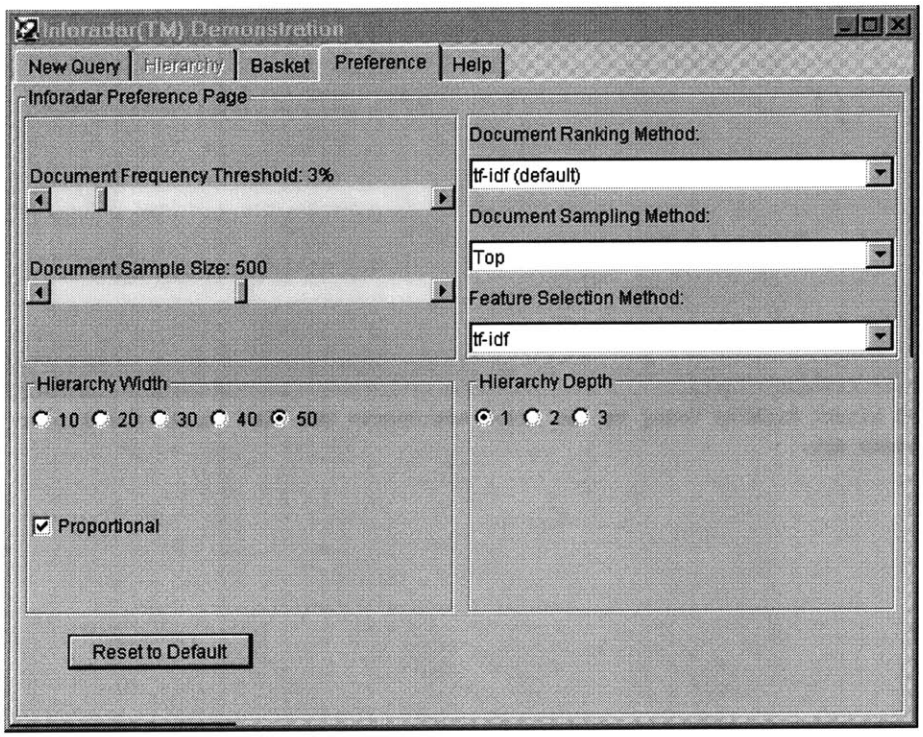



Figure 4-8: The Preference panel

The preference panel, shown in Figure 4-8, allows a user to configure a number of parameters used by the search engine to generate the query hierarchies. A change in the preference panel becomes active the next time a new query is entered or a node is expanded for the first time. The hierarchy width is the maximum number of child queries that any query may have. Hierarchy depth refers to the length of the longest path from

the selected query to a leaf query. The document frequency threshold parameter allows the user to control the maximum document frequency (i.e. the number of documents containing the term) allowed for terms selected as nodes. The document sample size controls the number of documents scanned for terms as well as the number of documents retrieved for the seed query.

As can be shown in the examples in this section, interactive query hierarchies provide a novel user interaction model that integrates browsing, searching and query formulation. The use of queries to specify categories avoids some of the semantic ambiguities associated with other system-assisted search result visualization approaches like agglomerative clustering. Queries also serve as extremely compact yet informative category descriptors. Users implicitly formulate new queries as they expand the query hierarchy. Applet persistence gives users complete freedom to navigate the hierarchy in any order without losing track of the information context provided by the results of previous user interface operations.

## 4.4 Interactive Query Hierarchy Generation

This section describes the algorithm used by *Inforadar* to generate interactive query hierarchies. Although *Inforadar* supports the extended Boolean query model described in Section 4.2, the initial description of the algorithm will remain applicable to any specific query model as long as it provides for document ranking. Later in the section we will present a number of optimizations specific to the *Inforadar* query model.

One simple way to generate a query hierarchy from a seed query is depicted in Figure 4-9. The algorithm generates a hierarchy in three steps: a *query refinement* step, a *query combining* step and a *query lookahead* step. The query refinement step extracts and selects refinement terms. The query combining step constructs child queries by combining the terms from the refiner with the seed query. Finally, query lookahead evaluates the result set of each child query. The process continues recursively by considering each child

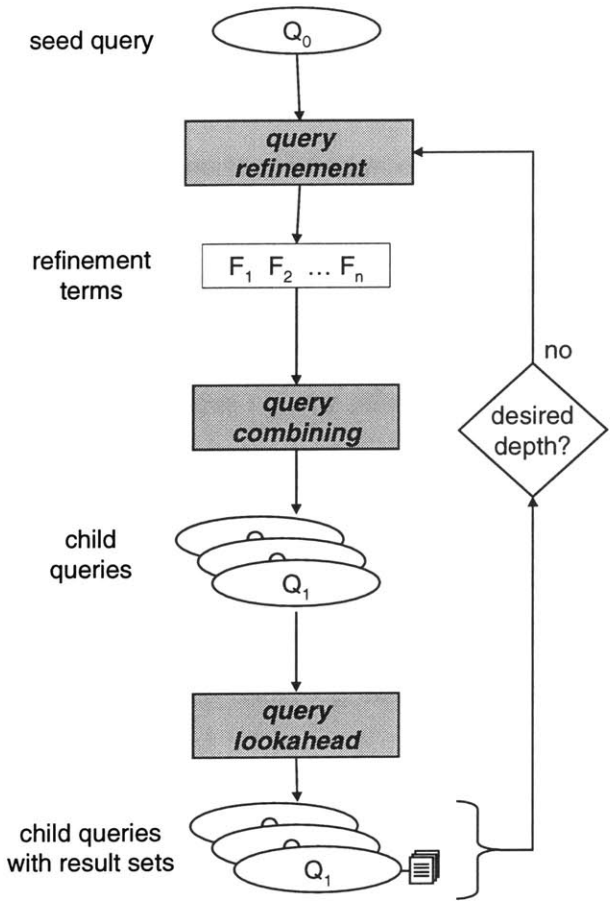



Figure 4-9: Abstract query hierarchy generation algorithm

query in turn as a new seed query until the desired hierarchy depth is reached. For clarity, the figure abstracts out several important details. For instance, the mechanism used to combine extracted terms with the seed query as well as how the desired depth is determined remain unspecified. The remainder of this section fills in these and other details of the algorithm.

There are a number of possible ways of combining query refinement terms with the seed query to build child queries. The possibilities are delimited by the query model. Under a query model supporting logical operators, a simple approach would be to generate child queries by conjoining each single refinement term with the seed query. The

result sets of such conjunctive child queries are subsets of the seed query's result set. As a result child queries do not have to be evaluated from scratch. Instead their result sets can be computed starting from the result set of the seed query. Another possibility is to combine the terms disjunctively, but this option requires that the inverted file entry for the new term be read into memory since it may include documents not in the result set of the seed query. Other more complicated approaches and combinations of approaches are also conceivable.

The first *Inforadar* prototype builds each child query by combining a single refinement term conjunctively with the parent query. The rationale for this design decision includes the following reasons:

- The approach is simple and we want to experiment with a simple approach before experimenting with more complicated ones.
- Child queries can be evaluated using the parent result set as a starting point.
- The coverage query refinement algorithms developed in Chapter 3 can be readily integrated into the query hierarchy generation algorithm.

We leverage the results of Chapter 3 by expanding the coverage query refinement algorithm developed in that chapter into a query hierarchy generation algorithm. The coverage query refinement algorithm uses query lookahead to eagerly compute the result sets of refined queries.

Although conjunctive queries may not be optimal in many circumstances the runtime efficiency gained by constraining the ways in which refinement terms can be combined with the seed query are substantial. Our experiments in Section 3.4 suggest that more complicated ways of building child queries may not yield practical query hierarchy generation algorithms.

Algorithm *IQH* in Figure 4-10 incorporates the *LOOK* and *CTS* algorithms introduced in Chapter 3 into a query hierarchy generation algorithm. The pseudo-code makes

Let
$q$ = root query
$w$ = hierarchy width
$d$ = hierarchy depth

(1) Evaluate the root query $q$ to obtain $\mathcal{D}(q)$

(2) If $d = 0$, STOP

(3) If $d > 0$, run *LOOK* to obtain a set of pairs $\{< (q \wedge f_i), \mathcal{D}(q \wedge f_i) > | f_i \in \mathcal{F}(q)\}$ of child queries and their result sets

(4) Run *CTS* to select an optimal subset of $w$ pairs $< q_i, \mathcal{D}(q_i) >$

(5) For each selected child query pair $< q_i, \mathcal{D}(q_i) >$ do

    (6) Recursively generate a hierarchy of depth $d - 1$ rooted at $q_i$ starting at step (2).

Figure 4-10: Algorithm *IQH* generates a multilevel query hierarchy

heavy use of the definitions introduced in Section 3.2. The notation $< q, \mathcal{D}(q) >$ is used to represent tuples.

*IQH* has the advantage that only the seed query needs to be evaluated from scratch. Still, the number of queries that must be processed is exponential in the depth of the tree, although the amount of work per query decreases with the depth of the hierarchy. Therefore, it is important that the amount of effort spent expanding the tree produces as much benefit for the user as possible. Ideally, if one could predict the portions of the tree of most interest to a user, then the algorithm could automatically attempt to expand these areas of interest. Unfortunately, it is often the case that the algorithm lacks the information necessary to make this prediction. The solution we adopt in *Inforadar* is to give the user absolute control of the areas of a query hierarchy that are expanded. As shown in Section 4.3, *Inforadar* generates a query hierarchy of depth one by default and allows the user to expand it by double-clicking nodes of interest. Users can also ask

*Inforadar* to generate deeper hierarchies using the Preference panel.

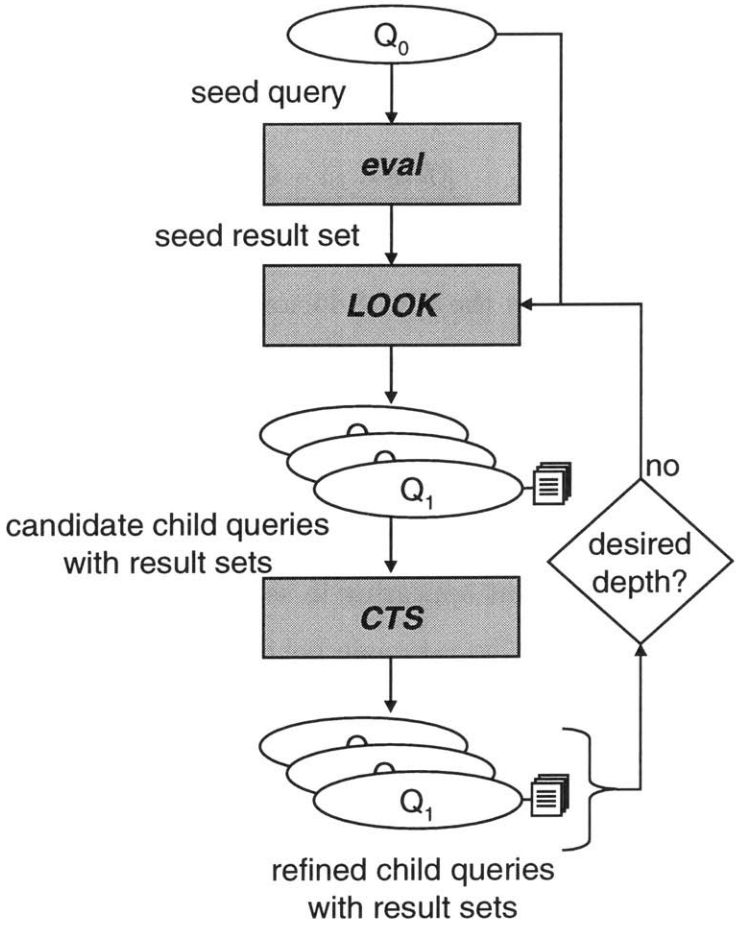



Figure 4-11: Flow diagram of the *IQH* query hierarchy generation algorithm

Figure 4-11 shows the flow diagram of the *IQH* algorithm. The eval module performs two steps: query evaluation and result set sampling. *LOOK* performs term extraction, query combining and query lookahead. Finally, *CTS* selects the best queries to display to a user.

After experimenting with our implementation of *IQH* we discovered that the algorithm often generated child queries for a given query that looked very similar to the children of its sibling queries, even though the corresponding queries looked very different. We discovered that this happened because *CTS* often selected terms appearing in

documents in the intersection between the result sets of the sibling queries.

Fortunately, we also discovered that as a side effect, *CTS* provides a mechanism for ameliorating this problem. In step (6) of Figure 3-7 *CTS* iteratively selects the term that maximizes the value of the expression:

$$\frac{\|\mathcal{D}(q \wedge f) - C\|}{\|\mathcal{D}(q \wedge f)\| + \|C\|}$$

In this expression, $C$ represents the set of documents covered by the terms that have been selected in previous iterations. Intuitively, the next term $f$ selected is the one whose corresponding result set has the most documents that do not contain any of the other terms previously selected. Let $A$ be the set $\mathcal{D}(q \wedge f) - C$ the documents that contain $f$ but not any other previously selected terms. We modified our algorithm to assign higher precedence to terms appearing in set $A$ when expanding the subhierarchy rooted at the node labeled $f$. The rationale behind this is that the set $A$ contains the documents that are more likely to be particularly well represented by term $f$ and not other different terms. We plan to conduct experiments to determine the efficacy of this approach in dealing with this type of term redundancy.

## 4.5 Design of the *Inforadar* Prototype System

The *Inforadar* search system is composed of three main subsystems: a search engine server, a client applet and an indexing subsystem. The client applet implements the graphical user interface (GUI) and sends queries to the search engine via the network. The search engine handles query requests from potentially many clients simultaneously. To process a request, the server consults a set of local data structures previously constructed by the indexing subsystem. In the remainder of this section we discuss the most important design decisions behind the communication protocol, the search engine and the indexing subsystems. The client applet was described in detail in Section 4.3.

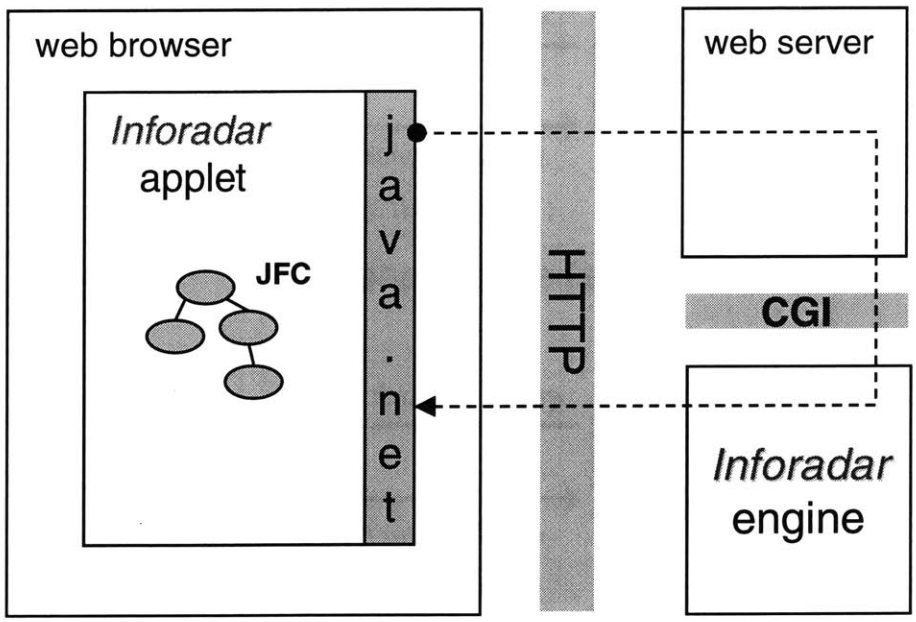



Figure 4-12: Software components in the *Inforadar* prototype

### 4.5.1 Communication protocol

Figure 4-12 illustrates the organization of the *Inforadar* client/server design. The search engine runs as a CGI [10] application that is triggered by a Web server. Communication between the client and the server is through a simple protocol implemented on top of HTTP [6]. Depicted in Figure 4-13, the protocol consists of a simple request response interaction triggered by the client. As is customarily done for this type of application, information is communicated between client and server by means of CGI variables encoded as part of the URL used to access the search engine.

What distinguishes the *Inforadar* protocol from other search engine protocols is the nature and amount of information communicated between the client and the server. When a query is first submitted (diagram (a) in Figure 4-13), the client simply sends the encoded query to the server. However, as we showed in Section 4.3, a user can expand queries by pointing and clicking with the mouse. In response to a GUI event to expand a query node (diagram (b) in Figure 4-13), the client not only sends the query to the server,

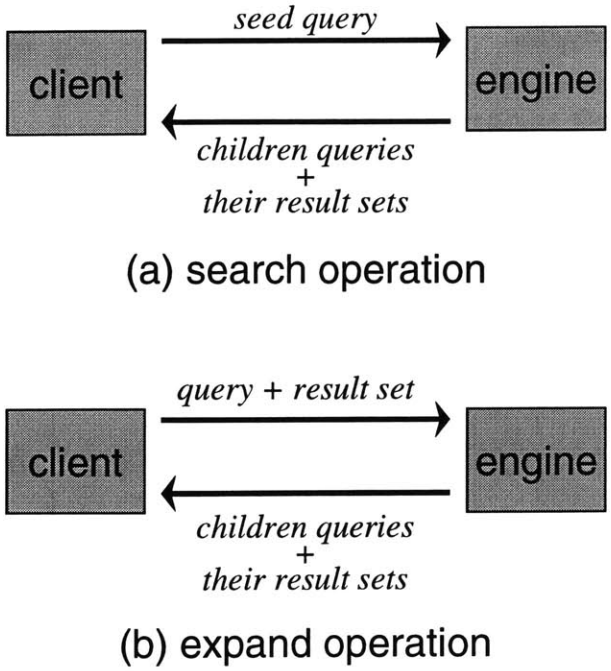



Figure 4-13: *Inforadar* client server protocol

but also the result set computed for the query in a previous request. This information is critical in preventing the search engine from having to compute the query from scratch and in maintaining the semantics of the interactive query hierarchy consistent across `expand` operations. In addition, as we saw at the end of Section 4.4, this result set contains additional information that is useful in improving the quality of the new level of child queries to be computed.

The search engine reply to both search and expand operations contains the same information. The response consists of a new set of child queries for the query submitted by the client, including the result set of each child query, as computed by the query lookahead algorithm. A simple compression algorithm makes sure that a list holding only one copy of the information (e.g. title, URL, abstract) associated with each result set document is returned. The result sets of child queries consist of pointers into this list. We plan to incorporate more sophisticated compression algorithms in future versions of the

prototype. In Section 4.6 we will show some measurements of the amount of information sent by the server in response to a query and show that transferring a substantially larger amount of information than the typical search engine does not play a major role in overall response time of our prototype search system.

### 4.5.2 Search and Refinement Engine

The *Inforadar* search engine comprises approximately ten thousand lines of C++ code [63]. The development environment consisted mostly of freely available GNU tools (www.gnu.org) including the G++ compiler, the GDB debugger and EMACS, all running under RedHat Linux version 5.1. Early in its development, the search engine made heavy use of the GNU libg++ library, but since the adoption of the C++ standard in 1998, we have ported the entire search engine to use standard C++ class libraries.

This section is not intended to provide a complete and fully detailed description of our engine implementation. Many of the techniques we use are fairly standard and well known. Instead, we will describe the more interesting and novel aspects of our design. Our implementation has been mostly driven by the desire to experiment with alternative information retrieval algorithms. With this in mind, our design exploits procedural abstraction to provide hooks allowing us to plug into our system, say, different term weighting or document ranking schemes without having to rewrite significant portions of the code.

One of the many interesting lessons that we learned during the implementation of *Inforadar* was that query evaluation and query refinement entail essentially the same computation. The process is illustrated in Figure 4-14. Several lists of objects are first retrieved from a database. Each list is filtered to determine which objects should be considered and which should be discarded. Every element of every filtered list is individually weighted. Then all the lists are merged into a single list. The weights of objects appearing in more than one list are combined. The resulting list is scanned to determine, based on the combined weights, which objects should be selected. The final

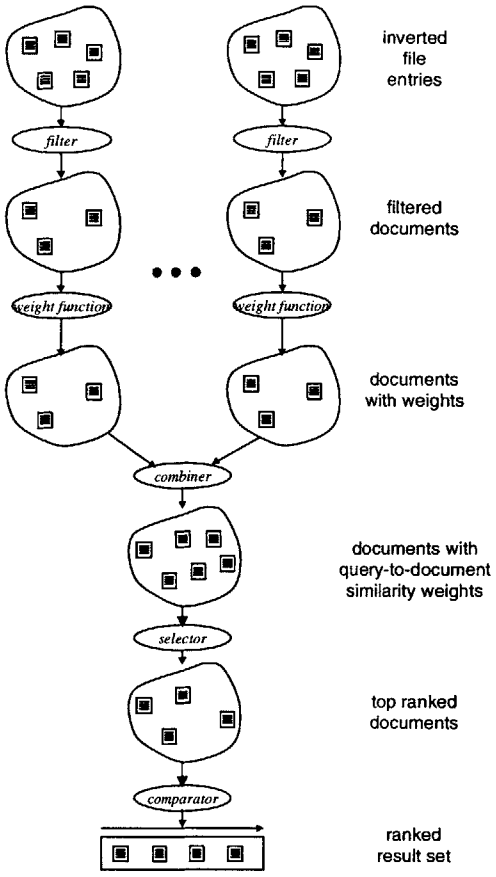



Figure 4-14: Use of procedural parameters in the *Inforadar* search engine.

list is then sorted by weight and returned as the result.

For query searching the initial lists are retrieved from an inverted file that maps each term in the corpus to the list of documents containing that term. Query refinement retrieves the initial lists from a file mapping documents to the terms that they contain. The specific filtering and weighting of the lists may be different, but both processes are in essence the same.

We devised a highly modularized design with an API consisting of five types of procedures that could be plugged in to customize the functionality of the search engine. The five types of procedures are *filters*, *weight functions*, *combiners*, *comparators*, and *selectors*. Selectors were introduced late in the development process after we started

experimenting with coverage term selection methods. Before then, term selection was hard-coded to select the terms with highest weights. Because of the nature of weight functions, algorithms such as $DM_{nfx}$ from Chapter 2 are limited to consider individual properties of terms. Selectors provide an abstraction barrier over a wider range of possible term selection methods, including coverage term selection.

We can further illustrate the use of the five procedural parameters in the *Inforadar* API with an example evaluation of a vector query with two terms: $f_1$ and $f_2$. As usual query evaluation entails generating a ranked list of documents matching the query. In this example we assume that the documents are ranked by the number of occurrences of a term within a document (tf). Initially, the search engine retrieves two lists of documents $l_1$ and $l_2$ from the inverted file for terms $f_1$ and $f_2$ respectively. One way to "plug in" the desired ranking method is to define the five procedural parameters as follows:

**Filter** Does not discard any documents.

**Weight Function** The weight of each document in list $l_i$ is simply the within-document term frequency of term $f_i$, typically available from the inverted file. Every document not present in a list $l_i$ is assigned a weight of zero in that list.

**Combiner** The weight of each document in the combined list $l'$ is the sum of the weights of the document in each of the lists $l_1$ and $l_2$.

**Comparator** Documents are sorted by their weights.

**Selector** Selects documents at the top of the sorted list.

Our modular design has proven general enough to allow us to implement all the term weighting, document ranking and term selection methods described throughout this dissertation. In addition, we were able to implement the *LOOK* algorithm using these five procedural parameters. *LOOK*'s initial lists come from the file that maps documents to terms. Its weight function initializes the result set of each refined query as each term is encountered. The combiner merges the result sets of equal terms appearing

in different documents. The *CTS* coverage term selection algorithm of Section 3.3 is implemented as a selector procedure.

### 4.5.3 Indexing Subsystem

To process queries, the search engine requires a collection of data structures holding information about the indexed documents. These data structures are generated by the *Inforadar* indexing module. The design of the *Inforadar* indexing module is quite typical. We present it in this section not because of its novelty, but rather, to provide the reader with details that are necessary to fully understand and appropriately interpret the experimental results presented throughout the dissertation.

The design of the *Inforadar* indexing module was primarily driven by ease and speed of implementation. Indexing was assumed to be carried out off-line. *Inforadar* evaluates queries using a static snapshot of the document corpus. An updated snapshot can be concurrently computed by the indexing module, but it does not become active until all the documents in the corpus have been processed.

Figure 4-15 illustrates a flow diagram of the indexing process as it goes through its four phases: *scanning*, *stopping*, *stemming* and *file inversion*. The figure shows the impact of each indexing phase on a hypothetical sample document numbered 10 containing the English phrase "Black Resistance Against the South African Government."

Our prototype implementation actually integrates scanning and stopping into a single software module. *Scanning* identifies and separates the terms in each document. In *Inforadar* any space-separated string of alphanumeric characters starting with a letter is considered to be a term. *Stopping* removes terms that are too common to be of any use for retrieval purposes. Examples of such terms include articles, prepositions and other extremely common terms like "different" or "nobody." The *Inforadar* indexing module reads a list of stop terms from a configuration file and integrates them into a finite state automaton that is then used to identify terms in the input text stream. The stop term list used by our prototype consists of about 400 common English terms.

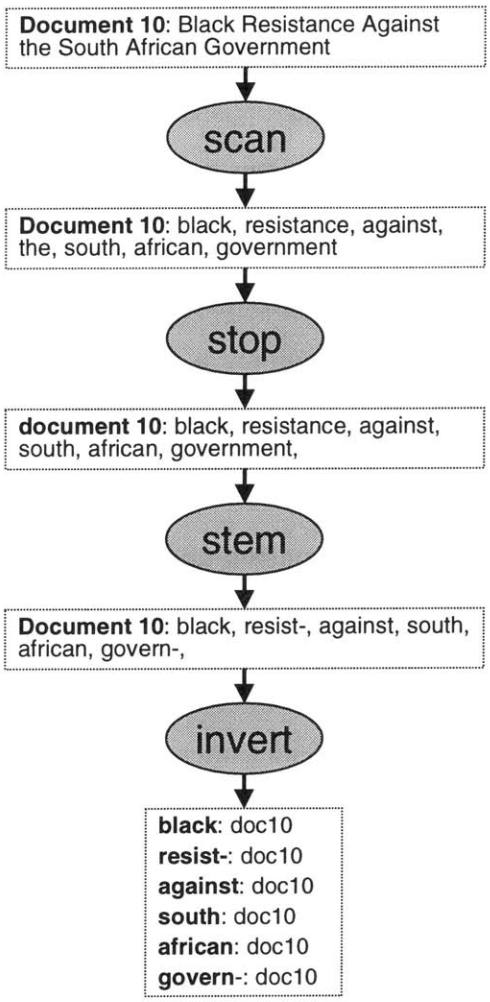



Figure 4-15: Indexing phases in the *Inforadar* indexing module.

*Stemming* detects minor variations of the same term and makes sure that only one variation of each term is indexed. *Inforadar* uses an implementation of the classical Porter [46] stemming algorithm. The algorithm applies a hard-coded series of common rules of thumb for suffix removal. The rules are only applicable to English language texts. *Inforadar* always remembers the first unstemmed version of each term and uses it as the external representation of the term. To avoid confusing the reader and at the same time simplify our discussion, we will continue to use the word "term" to refer to stems.

*File inversion*, the final indexing phase, generates a mapping from each term to the list

of documents that contain that term. *Inforadar* stores this mapping in a hashed file indexed by a unique numeric identifier assigned to each term. The current implementation uses the GNU GDBM library to manage hashed inverted files.

The output of the three phase indexing process consists of the following data structures, each stored as a separate file:

**Term Dictionary** Holds term information that is not local to any specific document where the term appears. Implements a two-way mapping between the stemmed character string representation of a term, and its uniquely assigned numeric identifier.

**Document dictionary** Holds information specific to each document including document length, title, URL, and abstract.

**Inverted file** Holds a map from each numeric term identifier to the list of documents that contain the term with that identifier.

**Document file** Holds a map from each document's unique numeric identifier to the list of terms appearing in that document.

The current implementation of the *Inforadar* search engine must load both the term and document dictionaries into memory before it can start processing queries. Future implementations will incorporate caching in order to limit the amount of memory required by the engine. This design decision is reasonable when process startup time does not play any role in query processing response time. However, the current search engine prototype, being based on CGI, is implemented as a stateless process. As a result, the search engine must load both the term and the document dictionary into memory on every client request. Future versions of the search engine will maintain the state of loaded data structures across requests. One potential solution is to interface the search engine to the web server using the Java Servlet API [16].

One novel aspect of the *Inforadar* indexing module implementation is the algorithm used to generate the term dictionary data structure. We designed *Inforadar* assuming

the term dictionary can be stored in main memory. Therefore, it was important to keep this data structure as small as possible.

Search engines often discard any information not necessary for query evaluation. For instance, many of these systems do not need to store the textual representation of terms. To speed up term comparisons, terms are internally represented using unique numeric identifiers. For display purposes, systems like *Inforadar* that support query refinement must be able to map numeric term identifiers back to their textual representation. In these systems the mapping provided by the term dictionary must be a two-way mapping.

A time efficient and conceptually simple approach to implement this two-way mapping can use two hash tables, one for each direction. However, this implementation is not space efficient since hash tables must store data sparsely in order to be effective and avoid collisions.

After observing that our search system imposes rather asymmetrical demands on the different mapping directions, we came up with a different approach. Mapping textual to numeric representation occurs once per query term, while mapping numeric to textual representation occurs once per refinement suggestion. Since typical queries are short (less than five terms) and often a query refinement algorithm displays several term suggestions per query, the system imposes higher demands on the speed of the numeric to textual direction.

Our solution stores the term dictionary as a table indexed by numeric term identifier. However, the numeric term identifiers are assigned so that their numeric ordering corresponds to the lexicographical ordering of their textual representations. This is not a trivial matter since, during indexing, terms are encountered in the order in which they appear in the input text stream.

We borrow a backpatching technique from compiler technology [2] to assign unique lexicographical identifiers to terms in two phases. Each term is assigned a temporary sequential identifier at the time it is first encountered in a document. A posting record containing the temporary term identifier and the number of occurrences in the current

document is appended to a postings file as the terms are encountered. After all the documents are scanned, the entire set of terms in sorted lexicographically and terms are assigned new unique identifiers in this order. The scanner saves a back-patching table mapping temporary sequential identifiers to the permanent lexicographical identifiers. The term dictionary file is stored at this time using the permanent term identifiers. However, the entries in the postings file remain with their temporary identifiers. During file inversion, the back-patching table is used to update the temporary term identifiers in the postings file. The final inverted file ends up using the lexicographically ordered identifiers.

The advantage of our approach is that we can look up the textual representation of a term in worst-case $\mathcal{O}(1)$ time. The reverse mapping is implemented in worst-case $\mathcal{O}(\log N)$ time, N being the size of the dictionary, using binary search. This level of performance is obtained without any data structure duplication.

## 4.6 Experiments

This section presents the results of two experiments conducted to measure the performance of the *Inforadar* network search system as described in Section 4.5. The first experiment measures the runtime efficiency of our query hierarchy generation algorithm. The second experiment is a pilot user study comparing *Inforadar* with a control system supporting a user interface more typical of current commercial search engine technology.

### 4.6.1 Performance Measurements

Figure 4-16 breaks down the average execution time consumed by *Inforadar* to generate 50 child queries for each of 15 test queries using three versions of the $IQH$ algorithm (Section 4.4). The versions differ in the term selection algorithm used. One version uses $DM_{nfx}$, from Chapter 2, the second uses $CTS$, from Chapter 3 and the third uses COMBO also from Chapter 3.

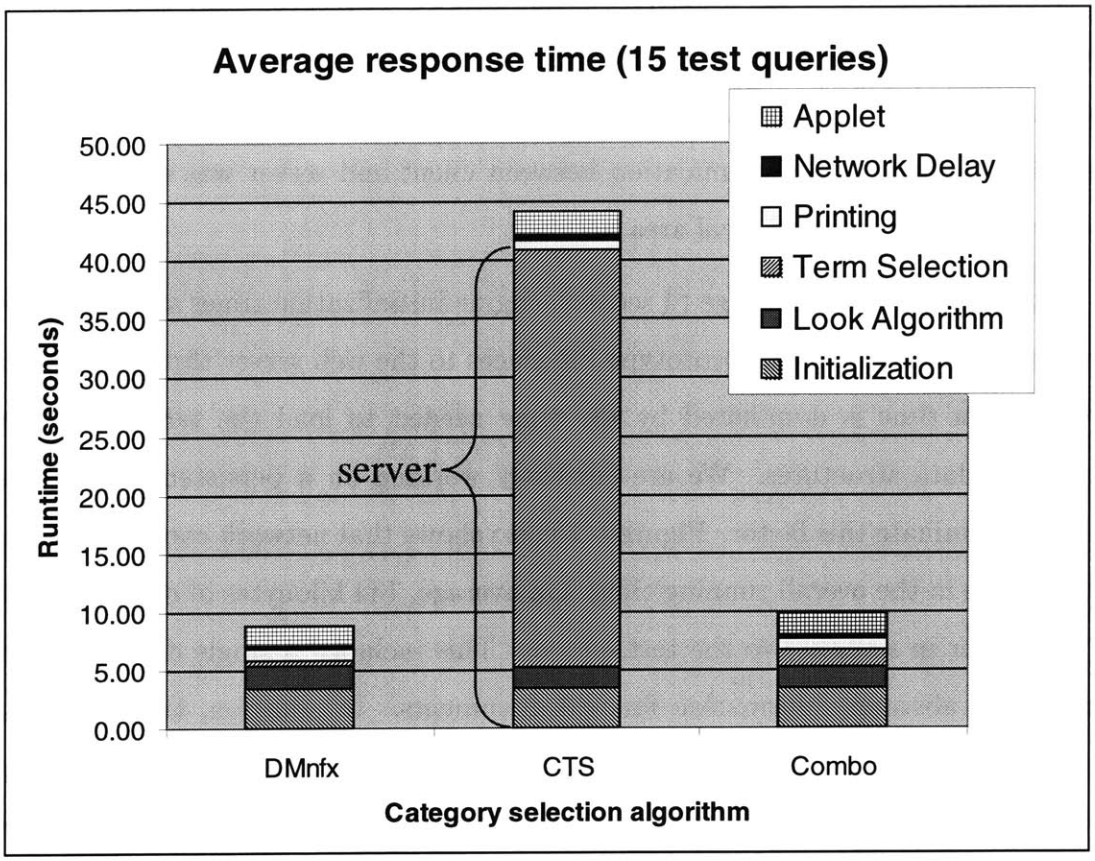



Figure 4-16: Runtime achieved by *Inforadar* generating 50 child queries

Since this experiment required manual submission of each test query, we computed runtime averages over a small random sample of 15 short TREC queries. The test queries were automatically generated from TREC topics using the same techniques used in Chapter 2. As in Chapter 2, we chose to analyze short queries because these are the most common queries submitted by real users [13]. It is important to keep in mind that even though each algorithm was requested to generate 50 child queries, $IQH$ must compute the result sets of a number of queries equal to the number of terms extracted from the documents in the result set sample. For instance, the average number of refined queries evaluated for the 15 test queries above was about 7,000 queries in our test corpus of 84,660 press articles.

As in Section 3.4.6, the tests were conducted on a virtually unloaded 400MHz dual Pentium PC running the Linux operating system (RedHat 5.2). The storage subsystem consisted of a 45 Gigabyte RAID level 5 disk directly connected to the PC via an ultra-wide SCSI interface. Communication between client and server was over a 10 megabit per second ethernet-based local area network.

The experiment shows large (4 second) engine initialization times stemming from the fact that the current engine prototype interfaces to the web server through CGI. Engine initialization time is dominated by the time needed to load the term and document dictionary data structures. We are currently working on a persistent server that will virtually eliminate this factor. Figure 4-16 also shows that network communication plays a minor role in the overall running time. On average, 144 kilobytes of data were returned by the server in response to the test queries. This included a single document list with header and abstract information for 500 documents. In addition, the reply from the search engine included 50 query hierarchy node labels and their corresponding result sets encoded as arrays of cross-references to the result set document list. No further data compression was performed.

The speed of the *LOOK* fast query lookahead algorithm is evidenced by the fact that, on average, the server was able to eagerly evaluate 7601 queries in 1.13 seconds. The results also show that our approximation algorithm for term selection takes a long time (35.52 seconds) to select 50 out of 7601 terms. Clearly, this component is the bottleneck of our query hierarchy generation algorithm. *COMBO* offers an attractive alternative achieving only 18% higher response times for the entire query refinement process over $DM_{nfx}$.

Algorithm *IQH* is capable of generating a single level query hierarchy in an amount of time that is within a constant factor of the time of previous term frequency-based query refinement algorithms [67]. These measurements confirm that query lookahead can be accomplished with the same asymptotic complexity as query refinement. This result by itself is important because it shows that any system that provides query refinement could

incorporate query lookahead at a small additional cost.

### 4.6.2 Pilot User Study

One of the best ways to test a new user interface idea is to test it on real users. We have conducted a small pilot study to assess the potential for improvement in retrieval effectiveness attainable by *Inforadar*. The control system to which *Inforadar* is compared supports a user interface more typical of what is available from Internet search service providers. This traditional user interface simply allows a user to submit queries manually and inspect their result sets sequentially. A number of previous user studies have influenced our own. We are particularly indebted to [7, 42, 66].

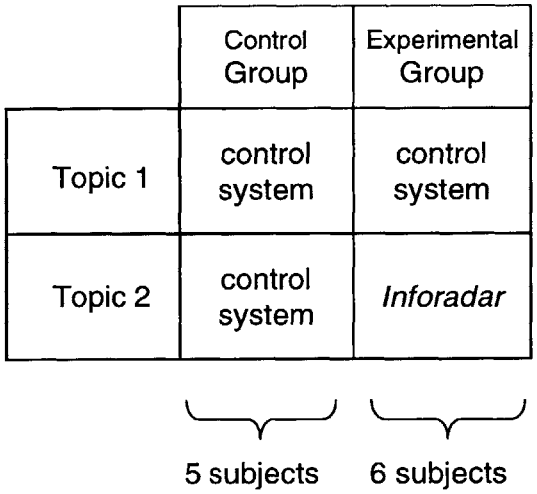



Figure 4-17: Format of the pilot user study

The group of eleven subjects consisted of MIT students reporting moderate experience with Internet search engines. Each student was paid $10 for the hour in which they received a short tutorial and conducted searches on two different topics. Figure 4-17 illustrates the format of the study. The group of eleven students was divided into two groups. A control group of five subjects performed both searches using the control system (without query hierarchies). An experimental group with the remaining six sub-

jects performed their first search using the control system and the second search using *Inforadar*. The description of each topic appears in Appendix A. The subjects were instructed to collect in the document basket (see Section 4.3) as many relevant documents as they could find in twenty minutes. The relevance of each document was judged by the subjects themselves. After the allotted time, each subject printed his/her document basket and completed a short survey. The questionnaire, which was designed using the guidelines mentioned by Schneiderman in [56], is included in Appendix A.

At first sight the format of the pilot study may seem more complicated than necessary. There are important reasons for this which we proceed to explain.

It may seem, for instance, that subjects could have been given a single topic to run on one of the systems. It is important not to make inferences about the average performance of users on a single query. We decided to let every user perform a search on the control system so that the improvement in effectiveness across topics could be measured for each user individually. Looking ahead at Table 4.1 we can see that the experimental group achieved higher precision (0.82) on topic 1 than the control group (0.79), showing that the subjects in the experimental group are better searchers. The improvement in precision achieved by the experimental group on topic 2 cannot, therefore, be attributed to the new user interface.

Letting each user run the same query two times was not a feasible simplification either since this may cause the performance of the second search to be overestimated due to the additional knowledge about the query, and the available data relevant to it, obtained during the first search. These effects are also known as order/practice effects.

Before we proceed to analyze the results, it is important to enumerate a number of important limitations of the study that may have caused the performance of *Inforadar* to be underestimated. First, at the time that the study was conducted *Inforadar* was at an early stage of testing and optimization. The system was configured to return query hierarchies of depth 2 by default. This caused the response time for every single hierarchy generation or expansion to be in the order of minutes. Another limitation of the study

was that users were much more familiar with the control system than with *Inforadar*. Most subjects reported at least moderate experience with such systems, while none of then had ever experienced *Inforadar* before. A third limitation was that the topics the were assigned to users were relatively easy to formulate as queries. We expect *Inforadar* to perform better in situations where formulating a specific query may be hard. Given these limitations it is evident that we could only have expected the control system to clearly outperform *Inforadar*.

| Metric | Control(5 subjects) | | Experimental(6 subjects) | |
|---|---|---|---|---|
| | Topic 1 Control | Topic 2 Control | Topic 1 Control | Topic 2 Inforadar |
| total retrieved | 33.2 (12.1) | 29.8 (15.9) | 26.4 (12.5) | 17.6 (7.3) |
| total relevant | 122 | 102 | 122 | 102 |
| relevant retrieved | 25.8 (8.5) | 20.6 (11.2) | 21.0 (8.7) | 12.6 (5.6) |
| queries selected | NA | NA | NA | 30.6 (23.3) |
| queries expanded | NA | NA | NA | 6.2 (6.8) |
| precision | 0.79 (0.03) | 0.69 (0.07) | 0.82 (0.1) | 0.71 (0.08) |
| recall | 0.21 (0.06) | 0.20 (0.11) | 0.17 (0.07) | 0.12 (0.05) |

Table 4.1: Comparison of retrieval performance achieved by *Inforadar* versus a control system without query hierarchies. Standard deviations in parentheses.

The numbers in Table 4.1 reflect averages across all the subjects in the corresponding group. One outlying subject was removed from the experimental group because it achieved unusually high performance with *Inforadar*. The total number of subject in both groups considered in the averages was the same (5). In computing this table, a document was considered "relevant" if it was marked relevant in the TREC collection and "retrieved" if it was considered relevant by a subject. Precision was calculated as the ratio of relevant retrieved to total retrieved documents and recall as the ratio of relevant retrieved to total relevant documents.

Perhaps the most important conclusion that we can draw from the above results is that, contrary to our expectations, the control system did not clearly outperformed *Inforadar*. Both groups of subjects achieved a decrease in both precision and recall on

the second topic. In terms of precision, the control group achieved 0.10 less precision in the second topic, while the experimental group achieved 0.11 less. In terms of recall the difference is bigger. The control group achieved 0.01 less recall while the experimental group achieved 0.05 less recall. Both measure indicate a slight decrease in performance by the experimental group. Standard difference of means test [4] (p=0.05) show that these differences are not statistically significant.

In the surveys, users expressed their preference for the interactive query hierarchy interface. The chief complaint was that the system was too slow. The fact that *Inforadar* was configured to return hierarchies of depth 2 was a critical contributor to the intolerably slow response time. A configuration returning query hierarchies of depth 1 would have lowered response time significantly and possibly allowing users to find more relevant documents in the allotted time. A user could always expand the hierarchy interactively along the areas of most interest. Most users found the *Inforadar* GUI very easy to use.

We learned a number of important lessons from this pilot study. First, users should execute both searches in random order to avoid order/practice effects. Otherwise an improvement in effectiveness on the search conducted second may be a result of users having more experience with the system during their second search. Second, future studies should consider other experimental tasks in addition to traditional query searching where *Inforadar* could be expected to be more helpful. For instance, we are considering asking users to find answers to a set of questions related with a topic as opposed to just finding documents that are relevant to a topic. We expect *Inforadar* to be most useful for tasks where formulating a specific query is difficult but our tests have not been designed with these types of tasks in mind.

## 4.7 Summary

In this chapter we introduced *interactive query hierarchies*, a new user interaction paradigm unique in its use of a query language to denote categories upon which documents

are automatically organized. An interactive query hierarchy is rooted at a seed query provided by a user. Each child query is a refined or more specific version of its parent. Child queries are refined by conjoining terms extracted from the documents that match the parent query. Thus, each level of queries in the hierarchy provides a progressively finer description of the available information.

The use of logical conjunction to generate refined queries allowed us to incorporate into the interactive query hierarchy generation process, the algorithms for query lookahead (*LOOK*) and coverage feature selection (*CTS*) that we developed in Chapter 3. The *LOOK* algorithm was critical in obtaining an interactive query hierarchy system of acceptable performance that could be deployed into a production environment.

As a vehicle for demonstrating that practical network search systems based on interactive query hierarchies are feasible and that such systems have the potential to improve information retrieval effectiveness, we developed the *Inforadar* prototype network search engine. This chapter provided complete descriptions of the main components of the *Inforadar* system: the client applet, the search and refinement engine, the network protocol and the indexing module.

When it first receives a seed query, *Inforadar* returns, by default, a single-level query hierarchy. A user can navigate and expand the hierarchy at will according to her/his interests. Users can examine the result sets associated with child queries without having to submit the child queries to the server. These result sets are eagerly computed by the query lookahead algorithm. Hierarchy expansion, however, requires communication with the search engine. In response to an **expand** operation, the client applet sends the query being expanded as well as its result set to the search engine. This is necessary to ensure that the semantics of the hierarchy remain consistent across expansions.

Interactive query hierarchies offer the following advantages over document clustering and human-generated category hierarchies:

- Interactive query hierarchies avoid the semantic ambiguities inherent to document clustering by using a query language to unambiguously determine the groups of

documents belonging to each category.

- A query language provides for very compact descriptions of category contents

- Interactive query hierarchies are dynamic and query specific. Due to the complexity of creating and maintaining classifications by hand, human-generated category hierarchies are often static and cannot adapt to user queries. It is often not possible, for instance, to obtained a hierarchical organization of the set of documents that match a particular user query.

We conducted two experiments with our *Inforadar* prototype. The first experiment measured the average response time of the system across a sample of 13 test queries from the TREC collection. The most important conclusion that we can draw from this experiment is that, even with little code optimization, our prototype system demonstrates that a system based on the idea of interactive query hierarchies is feasible with today's hardware/software technology. As we discussed in Chapter 3, our $CTS$ coverage term selection algorithm is slow and impractical for large document sets. However, our goal in this dissertation has been to demonstrate that coverage term selection algorithms are necessary to adequately deal with the problems of information loss and term redundancy discussed in Section 3.2.

The second experiment, a pilot user study, compared *Inforadar* to a control system providing a user interface more typical of current commercial network search engines. The control system allowed users to submit queries and examine their results sequentially. Although our sample of subjects was too small to yield any statistically significant results, we described a number of important lessons from this study that we plan to incorporate into a more comprehensive follow-on study that we will be conducting in the near future. It was interesting to observe that even though the version of *Inforadar* used in the study was considerably slower than the current version, and also considerably slower than the control system, the search effectiveness achieved by users of *Inforadar* was not significantly worse than that of users of the control system. Through surveys, users communicated

their frustration with the slow speed of the system but reported feeling comfortable with the system even after very little training.

In summary, although more studies will be necessary before we can assess the effectiveness of *Inforadar* and interactive query hierarchies with statistical significance, the result of this dissertation look promising. Interactive query hierarchies offer a balance between semantic clarity, implementability and ease of use that in our experience is unique and superior to previous approaches to system-assisted search result visualization.

# Chapter 5

# Conclusions

Previous work on query refinement lacked adequate means for empirically comparing the performance of alternative algorithms. This dissertation proposed concept recall and precision improvement as experimental metrics that can be used for this purpose. We have used these new metrics to compare the effectiveness of widely used algorithms like $DM_{nfx}$ against $RMAP$, a new and simple algorithm that statically precomputes query refinement information in order to reduce response time.

Based on the premise that computer processing capacity will continue to increase at the current rate at least for the foreseeable future, we have investigated one way to invest this expected computing capacity to improve search effectiveness. Our technique is called query lookahead. Query lookahead eagerly evaluates queries automatically refined from an initial seed query. We introduced a fast query lookahead algorithm named $LOOK$ that simultaneously accomplished term extraction and query lookahead. Both our theoretical analysis and our experiments demonstrate that query lookahead can be performed at a cost that is within a constant factor of the cost for query refinement alone.

Our research introduced two applications of query lookahead: coverage term selection and interactive query hierarchies. Coverage term selection comprises a collection of term selection algorithms that weight candidate terms based on their group properties. In particular, the goal of a coverage term selection algorithm is to minimize the amount

of redundancy and information loss in the selected group of terms. Based on this goal and using concepts from probability theory, we developed a model of ideal term selection (Section 3.2). Unfortunately, under our model, ideal term selection turns out to be NP-complete, a proof of which as provided in this chapter.

In response to this NP-completeness result, we developed $CTS$, a greedy approximation algorithm for coverage term selection and demonstrated that coverage term selection has the potential to generate more informative and less redundant terms than algorithms basing their selection decisions on individual term-frequencies.

Query lookaead enabled the development of a practical system based on *interactive query hierarchies*. The new user interface organizes a result set into a hierarchy of categories denoted by queries. The main idea is to label categories of documents with query fragments that enable the user and the system to engage in a dialog in a single consistent language framework. Interactive query hierarchies improve previous query refinement user interfaces in three ways:

- They provide a hierarchical classification of a seed result set that uses queries as classification categories.

- They provide instant access to the result sets generated by refined queries without requiring iterative re-submission of queries to a search engine.

- They allow a user to navigate the hierarchical classification in all directions while maintaining the entire context of the search session.

In summary, although more studies will be necessary before we can assess the effectiveness of *Inforadar* and interactive query hierarchies with statistical significance, the result of this dissertation look promising. Interactive query hierarchies offer a balance between semantic clarity, efficiency and usability that is unique and has the potential to improve search effectiveness over previous approaches to system-assisted search result visualization.

## Future directions

Our research leaves several questions unanswered. The following are examples of areas where further exploration would be desirable:

### Phrase recognition

Automatic phrase recognition technology automatically detects clusters of terms that together have a meaning that is significantly different from the meanings of its constituent terms. For instance, the phrase "information retrieval" carries a meaning that is more specific than the independent meanings of the words "information" and "retrieval".

Our current implementation of interactive query hierarchies does not incorporate phrase recognition. As a result, our algorithms only consider terms independently of the phrases that they may be part of. When natural phrases are broken apart, the importance of their constituent terms if often underestimated. Statistically speaking, measures that combine within-document term frequency with inverse document frequency often fail to recognize important terms because when considered independently these terms appear in more documents than when they appear as a phrase.

Interactive query hierarchies could benefit enormously from effective phrase recognition algorithms. Our experience with *Inforadar* taught us that one very common class of redundant terms is the terms resulting from the splitting of natural phrases. One look at the tables in Section 3.4.2 will be enough to convince ourselves of this problem. A very common type of phrase splitting occurs when proper names like "Nelson Mandela" (see Table 3.5) are separated into first and last names.

### Better coverage term selection algorithms

This dissertation demonstrates that information loss is a real problem that algorithms that only consider individual term frequencies cannot adequately cope with. Our *CTS* algorithm demonstrates that the problem can be reduced, albeit at great runtime cost. *COMBO* offers an alternative achieving less information loss reduction by much better

performance. However, our work only represents the beginning of the search for efficient coverage term selection algorithms.

### Clustering analysis

It would be interesting to compare and contrast the quality of the clusters formed by interactive query hierarchies with those formed by other bottom-up approaches like agglomerative clustering in Scatter/Gather [15]. Traditionally, measuring the quality of automatically formed document clusters has been considered a difficult problem. Recent results [65] show promise in this direction.

### User studies

The question of whether or not user interfaces based on interactive query hierarchies are useful in practice can only be properly answered by a comprehensive user study. Our anecdotal experience suggests that users felt confortable with the *Inforadar* GUI and our small user study suggests that the GUI can improve (at least slightly) search effectiveness. However, these results are not statistically significant due to our small sample sizes.

### User interface improvements

Several improvements to the *Inforadar* GUI have been suggested by users of the current prototype. One such improvement would be to combine the `New Query` and `Hierarchy` panels (see Figure 4-6) into a single panel. This would require replacing the display of the selected query at the top of the `Hierarchy` panel with an input text box which will then serve two purposes: displaying the selected query and allowing a user to type a new query.

# Appendix A

# Information Packet Provided to Subjects During the *Inforadar* Pilot User Study

# Bibliography

[1] IJsbrand Jan Aalbersberg. Incremental relevance feedback. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–22, Copenhagen, Denmark, June 1992.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Intermediate Code Generation*, chapter 8, pages 463–512. Addison-Wesley, 1985.

[3] Michelle Q. Wang Baldonado and Terry Winograd. Sensemaker: An information-exploration interface supporting the contextual evolution of a user's interests. In *CHI 97 Human Factors in Computing Systems*, pages 11–18, Atlanta, Georgia, March 1997.

[4] Arnold I. Barnett. 15.061 intermediate statistics: Course notes, Spring 1997.

[5] R. C. Barrett and E. J. Selker. Finding what i am looking for: An information retrieval agent. Technical Report RJ 9816, IBM Research Division, Almaden Research Center, San Jose, California, May 1994.

[6] Tim Berners-Lee. Hypertext transfer protocol. RFC 1945, May 1996.

[7] Giorgio Brajnik, Stefano Mizzaro, and Carlo Tasso. Evaluating user interfaces to information retrieval systems: A case study in user support. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 128–136, Zurich, Switzerland, August 1996.

[8] Chris Buckley, Gerard Salton, James Allan, and Amit Singhal. Automatic query expansion using SMART:TREC3. In Donna Harman, editor, *Proceedings of the Third Text Retrieval Conference (TREC-3)*, pages 69–80, Gaithersburg, MD, November 1994. NIST and ARPA.

[9] Mike Burrows. A library for indexing and querying text, 1997. Talk at the MIT Lab for Computer Science.

[10] The common gateway interface. World Wide Web Document. URL http://www.hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[11] James W. Cooper and Roy J. Byrd. Lexical navigation: Visually prompted query expansion and refinement. In *Proceedings of the 2nd ACM Conference on Digital Libraries*, pages 237–246, Philadelphia, PA, 1997.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[13] W. B. Croft, R. Cook, and D. Wilder. Providing government information on the internet: Experiences with THOMAS. In *Digital Libraries Conference DL95*, pages 19–24, Austin, Texas, June 1995.

[14] Douglass R. Cutting, David R. Karger, and Jan O. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *16th Annual International SIGIR'93*, pages 126–134, Pittsburgh, June 1993.

[15] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *15th Annual International SIGIR*, pages 318–329, Denmark, June 1992.

[16] James Duncan Davidson and Suzzane Ahmed. Java servlet specification, November 1998. Available online at www.javasoft.com.

[17] Sun Microsystems Javasoft Division. Java foundation classes. World Wide Web Document, 1998. http://www.javasoft.com/products/jfc/index.html.

[18] Alvin W. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill, New York, 1967.

[19] Andrzej Duda and Mark A. Sheldon. Content routing in networks of WAIS servers. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 124–132, Poznan, Poland, June 1994. IEEE.

[20] Efthimis N. Efthimiadis. A user-centered evaluation of ranking algorithms for interactive query expansion. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 146–159, Pittsburgh, PA USA, June 1993.

[21] William B. Frakes and Ricardo Baeza-Yates, editors. *Information Retrieval: Data Structures & Algorithms.* Prentice Hall, Englwood Cliffs, New Jersey, 1992.

[22] G. W. Furnas, T. K. Landauer, L. M. Gómez, and S. T. Dumais. The vocabulary problem in human-system communication. *Comm. ACM*, 30(11):964–971, November 1987.

[23] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[24] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, chapter Appendix 3. W.H. Freeman and Company, 1979.

[25] David K. Gifford. Polychannel systems for mass digital communication. *Comm. ACM*, 33(2), February 1990.

[26] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole. Semantic file systems. In *Thirteenth ACM Symposium on Operating Systems Principles*,

pages 16–25. ACM, October 1991. Available as *Operating Systems Review* Volume 25, Number 5.

[27] David K. Gifford, John M. Lucassen, and Stephen T. Berlin. An architecture for large scale information systems. In *10th Symposium on Operating System Principles*, pages 161–170. ACM, December 1985.

[28] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification.* Java Series. Computer and Engineering Publishing Group, 1996.

[29] David A. Grossman and Ophir Frieder. *Information Retrieval: Algorithms and Heuristics.* Kluwer Academic Publishers, 1998.

[30] Donna Harman. Towards interactive query expansion. In *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 321–331, Grenoble, France, June 1988.

[31] Donna Harman. Chapter 14: Ranking algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 363–392. Prentice Hall, Englwood Cliffs, New Jersey, 1992.

[32] Donna Harman. Relevance feedback revisited. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1–10, Copenhagen, Denmark, June 1992.

[33] Donna Harman. TIPSTER information retrieval text research collection. CDROM set, Linguistic Data Consortium, 1994.

[34] Marti A. Hearst. Tilebars: Visualization of term distribution information in full term information access. In *CHI 95 Human Factors in Computing*, pages 59–66, Denver, Colorado, May 1995.

[35] Marti A. Hearst and Chandu Karadi. Cat-a-cone: An interactive interface for specifying searches and viewing retrieval results using a large category hierarchy.

In *Proceedings of the 20th International ACM SIGIR Conference*, pages 246–255, Philadelphia, July 1997. ACM.

[36] Renato Iannella, Nigel Ward, Andrew Wood, Hoylen Sue, and Peter Bruza. Digital libraries and the open information locator project. Technical Report DSTC 34, Research Data Network Cooperative Research Centre, Resource Discover Unit, Queesland, Australia, 1996.

[37] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: A study of user queries on the web. *SIGIR Forum*, 32(1):5–17, Spring 1998.

[38] Brewster Kahle and Art Medlar. An information system for corporate users: Wide Area Information Servers. Technical Report TMC-199, Thinking Machines, Inc., April 1991. Version 3.

[39] David Karger, July 1999. Personal communication.

[40] Jon Kelinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[41] Steve Kirsch. Infoseek's experiences searching the internet. *SIGIR Forum*, 32(1):3–7, Spring 1998.

[42] Jürgen Koenemann and Nicholas J. Belkin. A case for interaction: A study of interactive information retrieval behavior and effectiveness. In *CHI 96 Human Factors in Computing Systems*, pages 205–212, Vancouver, B.C., Canada, April 1996.

[43] Mickael Lesk. "real world" searching panel at sigir 97. *SIGIR Forum*, 32(1):1–4, Spring 1998.

[44] Lucy Terry Nowell, Robert France, Deborah Hix, Lenwood S. Heath, and Eduard A. Fox. Visualizing search results: Some alternatives to query-document similarity. In

*Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 67–75, Zurich, Switzerland, August 1996.

[45] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional cllustering of english words. In *Proceedings of the 31st Annual Meeting of the ACL*, pages 183–90, 1993.

[46] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–37, 1990.

[47] Yonggang Qiu and H. P. Frei. Concept based query expansion. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, Pittsburgh, PA USA, June 1993.

[48] Salton, E. A. Fox, and E. Voorhees. Advanced feedback methods in information retrieval. *Journal of the American Society for Information Science (JASIS)*, 36(3):200–210, 1983.

[49] G. Salton, E. A. Fox, C. Buckley, and E. Voorhees. Boolean query formulation with relevance feedback. Technical Report TR 83-539, Department of Computer Science, Cornell University, Ithaca, NY, January 1983.

[50] G. Salton, E.A. Fox, and H. Wu. Extended boolean information retrieval. *Comm. ACM*, 26(11):1022–1036, November 1983.

[51] Gerard Salton. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[52] Gerard Salton. Another look at automatic text-retrieval systems. *Comm. ACM*, 29(7):648–656, July 1986.

[53] Gerard Salton. *Automatic Text Processing.* Addisson-Wesley, Reading, Massachusetts, 1988.

[54] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. Technical Report TR 88-898, Cornell University, February 1988.

[55] Bruce R. Schatz, Eric H. Johnson, and Pauline A. Cochrane. Interactive term suggestion for users of digital libraries: Using subject thesauri and co-occurrence lists for information retrieval. In *Proceedings of the 1st ACM Conference on Digital Libraries*, pages 126–133, Urbana, Ilinois, 1996.

[56] Ben Schneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, Reading, MA, 1998.

[57] Mark A. Sheldon. *Content Routing: A Scalable Architecture for Network-Based Information Discovery.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, October 1995.

[58] Mark A. Sheldon, Andrzej Duda, Ron Weiss, and David K. Gifford. Discover: A resource discovery system based on content routing. In *Proceedings of The Third International World Wide Web Conference*, Darmstadt, Germany, April 1995. Also in Computer Networks and ISDN Systems, Elsevier North Holland, 27(1995), pp. 953–972.

[59] Mark A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, Jr., and David K. Gifford. A content routing system for distributed information servers. Technical Report MIT/LCS/TR-578, M.I.T. Laboratory for Computer Science, June 1993.

[60] Mark A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, Jr., and David K. Gifford. Content routing for distributed information servers. In *Fourth International Conference on Extending Database Technology*, pages 109–122, Cambridge, England, March 1994. Available as Springer-Verlag LNCS Number 779.

[61] Craig Silverstein and Jan O. Pedersen. Almost-constant-time clustering of arbitrary corpus subsets. In *Proceedings of the 20th International ACM SIGIR Conference*, pages 60–66, Philadelphia, July 1997. ACM.

[62] A. F. Smeaton and C. J. van Rijsbergen. The retrieval effects of query expansion on a feedback document retrieval system. *The Computer Journal*, 26(3):239–246, August 1983.

[63] Bjarne Stoustrup. *The C++ Programming Language*. Addison-WEsley, third edition, July 1997.

[64] Franklyn Turbak and David K. Gifford. *Applied Semantics of Programming Languages*. Draft used for MIT 6.821 graduate course on Programming Languages, 1999.

[65] Shivakumar Vaithyanathan and Byron Dom. Model selection in unsupervised learning with applications to document clustering. In *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia, June 1999.

[66] Aravindan Veerasamy and Nicholas J. Belkin. Evaluation of a tool for visualization of information retrieval results. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 85–92, Zurich, Switzerland, August 1996.

[67] Bienvenido Vélez, Ron Weiss, Mark A. Sheldon, and David K. Gifford. Fast and effective query refinement. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, PA, July 1997.

[68] Ron Weiss, Bienvenido Vélez, Mark A. Sheldon, Chanathip Namprempre, Peter Szilagyi, and David K. Gifford. HyPursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In *Proceedings of the Seventh ACM Conference on Hypertext*, Washington, DC, March 1996.

[69] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Confer-*

*ence on Research and Development in Information Retrieval*, pages 4–11, Zurich, Switzerland, August 1996.

[70] Justin Zobel and Alistair Moffat. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, Spring 1998.

# InfoRadar User Interface Study

*Information for Volunteers*

Please complete the following information:

Gender: _____ Age: _____

| | | |
|---|---|---|
| MIT Affiliated? | Yes | No |
| Student? | Yes | No |

Thank you for volunteering for the user study of our InfoRadar search engine. We appreciate the time and effort that you are contributing to help us in our development of a better information retrieval system.

As a participant in our study, you will be asked to first participate in a short, informal tutorial session on our search engine. After this is completed, you will perform two search tasks on up to two different search engines. Your specific tasks and individual instructions are included later in this packet. *Please read the instructions on those pages carefully before beginning with your queries.* After the tasks are completed, you will be asked to complete a short 5-10 minute survey.

*Throughout this experiment, it is important to remember that it is the search engine that is being evaluated, not you.*

**Please the pages of this packet intact**, as it will be collected at the end of the experiment. If you have any questions now or during the time you are here, please feel free to ask. Thank you again for volunteering for our user study.

# InfoRadar User Interface Study

*Volunteer Consent Form*

I have volunteered on my own accord to participate in this experiment, and have been fully informed in advance of the expectations that are being placed upon me. I understand the process of the experiment and have been given the opportunity to ask any questions that I have. I also understand that I have the right to withdraw consent to the experiment at any time.

My signature below indicates that I affirm the above statements, and that my consent was given prior to my participation in this study.

_____ _____

Signature Date

# InfoRadar User Interface Study

*Instructions for Search Queries*

The following two pages contain your query questions. Your task is to search for as many relevant documents as possible on a given topic.

The relevance of documents is binary; that is, a document is either relevant or not relevant. The specifics regarding the relevance of documents to a certain topic is described on the individual topic pages. Please read this description carefully. *When you find a relevant document, be sure to put it into the document basket.*

You will have a total of 20 minutes to find as many documents as possible that are relevant to the given topic. Good luck, and have fun!

# InfoRadar User Interface Study

*Query #1*

**You have 20 minutes to complete this search.**

Find as many documents as possible relating to **Anti-smoking actions by the government**.

A relevant document will provide at least one example of actions proposed or taken by public authorities anywhere in the world to limit or discourage the human use of tobacco.

NOT relevant are private anti-smoking initiatives, unless they are traceable in origin to government action. Also NOT relevant are contradictory, pro-tobacco government actions, such as tobacco price support programs and export encouragement.

# InfoRadar User Interface Study

*Query #2*

**You have 20 minutes to complete this search.**

Find as many documents as possible relating to the **treatment of Acquired Immune Deficiency Syndrome (AIDS) or AIDS Related Complex (ARC).**

To be relevant, a document must include a reference to at least one specific potential AIDS or ARC treatment.

# InfoRadar User Interface Study

*Post-Experimental Survey Questions*

1. How would you rate your past experience with other document search engines?

| No experience | | | | | | Very experienced |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

2. What were your overall reactions to InfoRadar?

| Negative | | | | | | Positive |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| Frustrating | | | | | | Gratifying |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| Difficult | | | | | | Easy |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

3. Describe your overall impression of the interactive query hierarchies.

| Redundant | | | | | | Precise |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| Not helpful | | | | | | Very helpful |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| Illogical | | | | | | Logical |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| Frustrating | | | | | | Gratifying |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

4. How helpful were the colored buttons in the hierarchical structure?

| Not helpful | | | | | | Very helpful |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

5. How helpful was the result set graph in finding relevant documents?

| Not helpful | | | | | | Very helpful |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

6. How would you rank the ease of use of InfoRadar?

| Very difficult | | | | | | Very easy |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

7. How would you describe the speed of the system?

| Too slow | | | | | | Adequately fast |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

8. How does InfoRadar compare with other search engines you have used?

| Much worse | | | | | | Much better |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

9. InfoRadar will be more effective than search engines like AltaVist.. for your task.

| Strongly disagree | | | | | | Strongly agree |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

10. Query hierarchies helped me make sense of large and imprecise result sets.

| Strongly disagree | | | | | | Strongly agree |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

11. The global view of the result set provided by the result set graph helped me determine which queries in the hierarchy to examine first.

| Strongly disagree | | | | | | Strongly agree |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

12. What qualities of InfoRadar make it better or worse than other search engines?

13. What improvements would you make to the InfoRadar system?

14. Use this space to provide us with any other comments or observations that you have.